

PenPoint™ User Interface Design Reference



PenPoint

PenPoint™



GO CORPORATION

GO TECHNICAL LIBRARY

.....

PenPoint Application Writing Guide provides a tutorial on writing PenPoint applications, including many coding samples. This is the first book you should read as a beginning PenPoint applications developer.

PenPoint Architectural Reference Volume I presents the concepts of the fundamental PenPoint classes. Read this book when you need to understand the fundamental PenPoint subsystems, such as the class manager, application framework, windows and graphics, and so on.

PenPoint Architectural Reference Volume II presents the concepts of the supplemental PenPoint classes. You should read this book when you need to understand the supplemental PenPoint subsystems, such as the text subsystem, the file system, connectivity, and so on.

PenPoint API Reference Volume I provides a complete reference to the fundamental PenPoint classes, messages, and data structures.

PenPoint API Reference Volume II provides a complete reference to the supplemental PenPoint classes, messages, and data structures.

PenPoint User Interface Design Reference describes the elements of the PenPoint Notebook User Interface, sets standards for using those elements, and describes how PenPoint uses the elements. Read this book before designing your application's user interface.

PenPoint Development Tools describes the environment for developing, debugging, and testing PenPoint applications. You need this book when you start to implement and test your first PenPoint application.

PenPoint™

PenPoint™ User Interface Design Reference



GO CORPORATION

GO TECHNICAL LIBRARY

Addison-Wesley Publishing Company

Reading, Massachusetts ♦ Menlo Park, California ♦ New York
Don Mills, Ontario ♦ Wokingham, England ♦ Amsterdam
Bonn ♦ Sydney ♦ Singapore ♦ Tokyo ♦ Madrid ♦ San Juan
Paris ♦ Seoul ♦ Milan ♦ Mexico City ♦ Taipei

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial capital letters.

The authors and publishers have taken care in preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Copyright ©1991–92 GO Corporation. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

The following are trademarks of GO Corporation: GO, PenPoint, the PenPoint logo, the GO logo, ImagePoint, GOWrite, NoteTaker, TableServer, EDA, MiniNote, and MiniText.

Words are checked against the 77,000 word Proximity/Merriam-Webster Linguibase, ©1983 Merriam Webster. ©1983. All rights reserved, Proximity Technology, Inc. The spelling portion of this product is based on spelling and thesaurus technology from Franklin Electronic publishers. All other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations.

PenTOPS Copyright ©1990–92, Sitka Corporation. All rights reserved.

Warranty Disclaimer
and Limitation of
Liability

GO CORPORATION MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT, REGARDING PENPOINT SOFTWARE OR ANYTHING ELSE. GO Corporation does not warrant, guarantee, or make any representations regarding the use or the results of the use of the PenPoint software, other products, or documentation in terms of its correctness, accuracy, reliability, currentness, or otherwise. The entire risk as to the results and performance of the PenPoint software and documentation is assumed by you. The exclusion of implied warranties is not permitted by some states. The above exclusion may not apply to you.

In no event will GO Corporation, its directors, officers, employees, or agents be liable to you for any consequential, incidental, or indirect damages (including damages for loss of business profits, business interruption, loss of business information, cost of procurement of substitute goods or technology, and the like) arising out of the use or inability to use the documentation or defects therein even if GO Corporation has been advised of the possibility of such damages, whether under theory of contract, tort (including negligence), products liability, or otherwise. Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitations may not apply to you. GO Corporation's total liability to you for damages from any cause whatsoever, and regardless of the form of the action (whether in contract, tort [including negligence], product liability or otherwise), will be limited to \$50.

U.S. Government
Restricted Rights

The PenPoint Software Developers' Kit documentation is provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in FAR 52.227–19 (Commercial Computer Software—Restricted Rights) and DFAR 252.227–7013 (c) (1) (ii) (Rights in Technical Data and Computer Software), as applicable. Manufacturer is GO Corporation, 919 East Hillsdale Boulevard, Suite 400, Foster City, CA 94404.

ISBN 0–201–60858–8

123456789–AL–9695949392

First Printing, April 1992

PenPoint User Interface Design Reference

▀ Foreword

Many people today share a vision of a new class of mobile, pen-based computers that will bring the power of computers to people in a more natural way.

The enabling technologies are now in place to realize this vision. Hardware technologies include lower-power displays and microprocessors, lighter weight, longer lasting batteries, and wireless networks. And now, with the introduction of PenPoint, the software foundation is in place: a multi-tasking, object-oriented operating system and application framework optimized to work with the pen.

But there is one essential element missing: the applications that will put all this technology to work for people in their daily lives.

No one knows yet what form the compelling pen-centric applications will take. Just as the introduction of graphic displays and the mouse in the 70s and 80s created a rich space of design possibilities that resulted in new classes of applications, the advent of pen-based computers is opening up a new design horizon for the 90s.

The discipline of user interface design has a key role to play in exploring this new horizon. We invite you to join us in the exciting challenge of making the vision of pen-based computing a reality!

Tony Hoeber

Chief User Interface Architect, GO Corporation

▼ Preface

▼ Audience and Purpose of This Manual

This manual is comprised of two separate books: Book I, *The PenPoint User Interface* and Book II, *Designing a PenPoint Application*.

▼▼ Book I: The PenPoint User Interface

Book I provides a complete overview of the user-level features of the PenPoint™ operating system. It takes a user's-eye-view of the system, describing the many screens, menus, notes, and sheets that the user sees while interacting with the system.

The information will be of particular interest to hardware manufacturers contemplating building pen computers or software developers contemplating writing pen-centric applications.

It will also be of interest to a wider audience, including anyone who wants to understand the newly emerging market in mobile pen computers.

▼▼ Book II: Designing a PenPoint Application

Book II gives guidelines for designing PenPoint applications. It assumes familiarity with the material covered in Book I.

It is addressed specifically to application designers, software engineers, and human factors specialists who are developing PenPoint applications.

▼ Summary of Contents

▼▼ Book I. The PenPoint User Interface

Part 1. Overview. Begins by describing the new class of users and devices and summarizing the operating system features that address them. Against that background, provides an overview of the main concepts and features of the user interface.

Part 2. User Interface Components. Describes the components provided by the PenPoint User Interface Toolkit for application developers. Includes controls (such as fill-in fields, choice lists, and buttons) and the containers they appear in (such as menus, option sheets, and error notes).

- Part 3. Standard User Interface Features.** Covers the features implemented by the PenPoint Application Framework and common to all applications, including standard menus and option sheets, standard commands such as Print, Send, Find, and Spell, the drag & drop interface for moving and copying, icons, and reference buttons.
- Part 4. System Applications.** Covers the notebook table of contents as well as special-purpose notebooks such as the Help notebook, the Settings and Connections notebooks, and the In box and Out box.
- Part 5. Bundled Applications.** Describes MiniText, a simple word processor, and MiniNote, a simple ink processor.

Book II. Designing a PenPoint Application

- Part 1. Design Background.** Gives a brief review of graphical user interface (GUI) basics. Gives a brief orientation to designing applications for mobile, pen-based computing.
- Part 2. Application Architecture.** Presents guidelines on the overall structure of the application and how it is presented to the user. Discusses the different types of applications: documents within a notebook, pop-up accessories, separate notebooks, and services. Also covers installation, managing embedded objects, and issues related to scalability.
- Part 3. User Interface Building Blocks.** Describes how to use the building blocks, including controls such as buttons, fields and lists, and various containers for controls, such as menus, dialog sheets, option sheets, and notes. Also covers guidelines for designing icons and writing Quick Help.
- Part 4. Processing Pen Input.** Covers issues in gesture processing, handwriting translation, and the use of pen strokes without translating. Also covers how to present input modes to the user.
- Part 5. Viewing and Manipulating Application Objects.** Covers issues related to viewing the applications data, including scrolling and zooming, and layout-related issues. Also presents detailed guidelines for selecting, moving, and copying application objects.

Related Documentation

GO Corporation provides complete suites of documentation for end users, hardware manufacturers, and software developers.

PenPoint user documentation includes:

- ◆ *Getting Started with PenPoint*. A brief introduction to using PenPoint.
- ◆ *Using PenPoint*. A more complete guide to using PenPoint.
- ◆ *Using GOWrite*. A booklet about GOWrite handwriting software.

PenPoint software developer documentation includes:

- ◆ *PenPoint Development Tools*. Covers SDK (Software Development Kit) installation and software development tools.
- ◆ *PenPoint Application Writing Guide*. Covers PenPoint architecture, writing applications, and sample code.
- ◆ *PenPoint Architectural Reference, Volumes I and II*. Detailed information on each of the subsystems in the PenPoint operating system.
- ◆ *PenPoint API Reference Manual*. “Datasheets” on all the functions and messages comprising the API (Application Programming Interface) to PenPoint.

Acknowledgements

Many PenPoint application developers provided valuable review and feedback to the guidelines in Book II. Special thanks to Arnold Blinn, Dan Bricklin, David Dunham, Normal Guadagno, Matt Kursh, Bill Lynch, Mark Moore, David Reed, Scott Schwartz, and Keith Wales.

PENPOINT USER INTERFACE DESIGN REFERENCE

BOOK 1 / THE PENPOINT USER INTERFACE

Part 1 / Overview

Chapter 1 / New Technology for New Users	9
Chapter 2 / Overview of the PenPoint User Interface	12

Part 2 / User Interface Components

Chapter 3 / Gestures	23
Chapter 4 / Controls	28
Chapter 5 / Menus, Sheets, and Notes	40

Part 3 / Standard User Interface Features

Chapter 6 / Standard Menus, Option Sheets, and Commands	55
Chapter 7 / Drag & Drop Interface	68
Chapter 8 / Icons	73
Chapter 9 / Reference Buttons	79
Chapter 10 / Busy Clock	81

Part 4 / System Applications

Chapter 11 / Notebook Table of Contents	85
Chapter 12 / Settings Notebook	90
Chapter 13 / Connections Notebook	105
Chapter 14 / Stationery Notebook	120
Chapter 15 / In Box and Out Box	121
Chapter 16 / Help	123

Part 5 / Bundled Applications

Chapter 17 / MiniText	129
Chapter 18 / MiniNote	135

Part 1 / Design Background

- Chapter 1 / UI Design Basics 149
- Chapter 2 / Designing for Mobile, Pen-Based Computing 151

Part 2 / Application Architecture

- Chapter 3 / Documents, Accessories, and Services 157
- Chapter 4 / Installation 162
- Chapter 5 / Managing Embedded Objects 166

Part 3 / User Interface Building Blocks

- Chapter 6 / Controls 175
- Chapter 7 / Menus 192
- Chapter 8 / Option Sheets and Dialog Sheets 200
- Chapter 9 / Status, Warning, and Error Feedback 210
- Chapter 10 / Help 215
- Chapter 11 / Designing Application Icons 216
- Chapter 12 / Putting the Building Blocks Together 223

Part 4 / Processing Pen Input

- Chapter 13 / Processing Gestures 231
- Chapter 14 / Processing Handwriting 241
- Chapter 15 / Presenting Input Modes 244
- Chapter 16 / Using Ink 254

Part 5 / Viewing and Manipulating Application Objects

- Chapter 17 / Scrolling and Zooming 263
- Chapter 18 / Scaling and Resizing 268
- Chapter 19 / Optimizing Layout for Small Screens 274
- Chapter 20 / Selecting 278
- Chapter 21 / Moving and Copying 285
- Appendix A / PenPoint Design Checklist 295

Book I: The PenPoint User Interface

PENPOINT USER INTERFACE DESIGN REFERENCE

BOOK 1 / THE PENPOINT USER INTERFACE

Part 1 / Overview

Chapter 1 / New Technology for New Users	9
Chapter 2 / Overview of the PenPoint User Interface	12

Part 2 / User Interface Components

Chapter 3 / Gestures	23
Chapter 4 / Controls	28
Chapter 5 / Menus, Sheets, and Notes	40

Part 3 / Standard User Interface Features

Chapter 6 / Standard Menus, Option Sheets, and Commands	55
Chapter 7 / Drag & Drop Interface	68
Chapter 8 / Icons	73
Chapter 9 / Reference Buttons	79
Chapter 10 / Busy Clock	81

Part 4 / System Applications

Chapter 11 / Notebook Table of Contents	85
Chapter 12 / Settings Notebook	90
Chapter 13 / Connections Notebook	105
Chapter 14 / Stationery Notebook	120
Chapter 15 / In Box and Out Box	121
Chapter 16 / Help	123

Part 5 / Bundled Applications

Chapter 17 / MiniText	129
Chapter 18 / MiniNote	135

▼ Introduction

▼ User Interface Design and Development Process

The PenPoint™ operating system has been in development for more than four years. The design of the user interface has been an iterative process involving close cooperation among many groups, both inside and outside of GO.

The important milestones in the development of PenPoint include:

- ◆ September 1987. GO founded.
- ◆ June 1988. "Proof of concept" demo.
- ◆ June - December 1988. "Strawman" user interface design.
- ◆ January - April 1989. Paper prototype testing.
- ◆ June 1989 - February 1991. Eight releases (every 2-3 months).
- ◆ February 1991. Developer's Release (runs on GO's 286-based prototype hardware).
- ◆ February - October 1991. PenPoint ported to 386 processor.
- ◆ October 1991. 386-based system shown at COMDEX.
- ◆ April 1992. Release 1.0.

▼ Hardware Requirements

The PenPoint operating system was designed to be portable and device-independent. The device-dependent code has been formalized into a Machine Independent Layer (MIL).

PenPoint 1.0 runs on the Intel 386 processor.

Minimum hardware requirements include:

- ◆ 4 or more levels of grey.
- ◆ Ink plane.
- ◆ Pen proximity sensing.
- ◆ The minimum display size is 320 x 200 pixels. The minimum display resolution is 72 pixels per inch.

Part 1

Overview

THE PENPOINT USER INTERFACE

PART 1 / OVERVIEW

■ Chapter 1 / New Technology for New Users	9
New Users	9
New Devices	10
A New Kind of Operating System	10
A New Kind of User Interface	11
■ Chapter 2 / Overview of the PenPoint User Interface	12
Basic Screen Layout: Notebook & Bookshelf	12
The Bookshelf	13
The Notebook	14
Paper as a Model for Presenting Information	14
The Notebook as a Model for Structuring Information	14
The Notebook as a Model for Accessing Information	15
Gestures	15
Dual Command Path	15
Core Gestures	16
Embedded Document Architecture (EDA)	17
Compound Documents	18
Borders and Controls for Embedded Documents	19
Creating Embedded Documents	20
Reference Buttons	20

Chapter 1 / New Technology for New Users

This chapter contrasts the current generation of personal computers and the newly emerging mobile, pen-based computers in terms of users, devices, operating systems, and user interfaces. Comparisons are presented in the form of four tables:

- ♦ **Users.** Knowledge workers versus interactive professionals.
- ♦ **Devices.** PC or laptop with mouse and keyboard versus pen-based tablet or palmtop.
- ♦ **Operating Systems.** Procedural with static connectivity versus object-oriented with deferred connectivity.
- ♦ **User Interfaces.** Graphical user interface (GUI) with desktop metaphor versus gestural user interface with notebook metaphor.

▀ New Users

Users: Knowledge Workers versus Interactive Professionals

	Knowledge Workers	Interactive Professionals
Skills and Background	Able and willing to type. Familiar with computer concepts.	May not type. May not be familiar with or have patience for computer concepts.
Focus of Attention	Concentrating on the task at hand for relatively long periods without interruption.	Interacting with others. Liable to be interrupted at any time. People, not the computer, are the main focus.
Usage Pattern	Longer work periods. Can tolerate setup and close-down time.	Need to consult the computer "on the fly."
Location and Posture	Sitting at a desk or table.	Variety of positions as the user moves around: sitting at a desk, sitting away from a desk, standing, walking, or lying down.
Social Setting	Alone, or in an office setting in which machine noise is acceptable.	Interacting with others: an interview with one person, a meeting with a group, working in a public setting such as an auditorium or classroom, where clacking keys and whirring disks are not acceptable.

New Devices

Devices: Desktop PCs versus Mobile Pen-Based Computers

	Desktop PCs	Mobile Pen-Based Computers
Cost	\$1,000 - \$10,000	\$500 - \$5,000
Form Factor	Desktop and laptop.	Tablet and hand-held.
Location	On the desktop.	Accompany the user; can be used anywhere.
Display	VGA (640 x 480 pixels)	Range of screen sizes and resolutions.
Connectivity	Connections are static and stable.	Can't assume a network connection always there. Connections come and go as user moves around. User goes to different sites with different types of networks.

A New Kind of Operating System

Operating Systems: Deskbound & Disk-Based versus Mobile & Scalable

	Current Systems	PenPoint
Memory Usage	Require three copies of code: one in memory, another on disk, and a third in a swap file.	Designed with low memory consumption in mind. Object-oriented design facilitates code-sharing.
On/Off	Reboot required after power-down. System returns to startup state.	Instant on—system returns to previous state immediately, without need to reboot.
Output	User interface is tied to VGA.	User interface is scalable to different display devices. Device-independent coordinate system and constraint-based layout allow applications to present themselves regardless of screen size, resolution, or aspect ratio, without being rewritten.
Network Connections	Disconnection is an error condition. Requires reboot to install new network protocol.	Will not lose data when sessions are interrupted by interference on wireless networks. PenPoint suspends and resumes the connection as the machine moves out of and back into range of the network. Connecting and disconnecting from network, or installing new network protocols, does not require rebooting. Several protocols can be installed at the same time, so user can switch between different networks by simply plugging in the relevant cables.

Operating Systems: Deskbound & Disk-Based versus Mobile & Scalable (Continued)

Data Transfer (Send, Receive, Print)	User can only send, receive, and print when connected to the network, modem, or printer.	User can issue commands such as Print, Fax, or Email at any time, without worrying about whether the computer is currently connected to a network, and count on the system to actually complete the operation whenever the appropriate connection is established.
---	--	---

▼ A New Kind of User Interface

User Interfaces: Desktop & Mouse versus Notebook & Pen

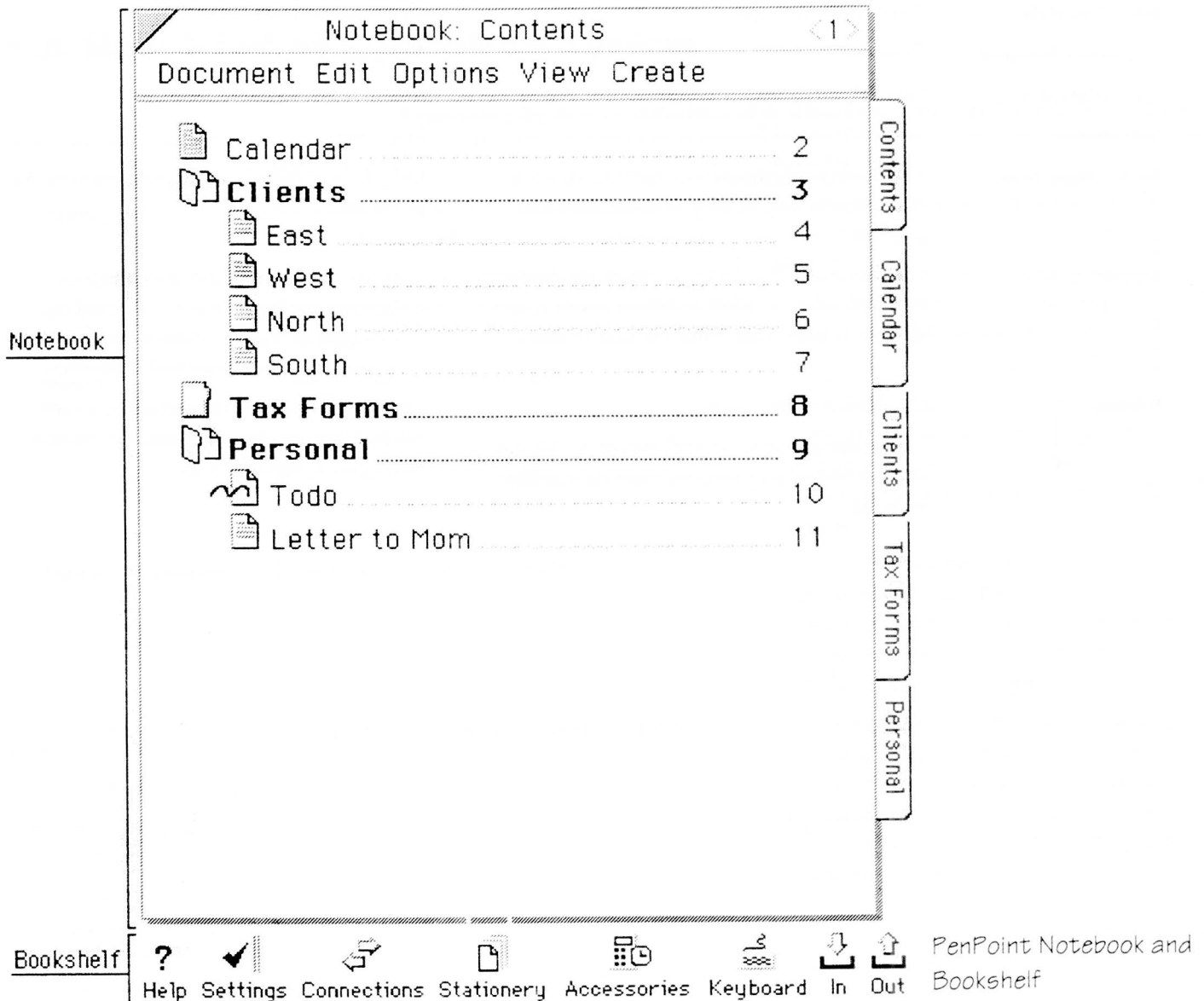
	Current PC Interfaces	PenPoint Interface
Input	Mouse and keyboard (pen optional).	Pen (keyboard optional).
Interaction Style	Point, click, and type.	Tap, flick, and write.
Organizing Metaphor	Desktop.	Notebook.
Visual Metaphor	Control panel.	Paper.
Data Model	Applications and files.	Documents.
Data Transfer Model	Two models: clipboard with cut/copy/paste for most objects; drag & drop to move icons and graphics.	Drag & drop model to move and copy objects of any type.
Combining data	User can create "hot links" in which the display in one application is linked to data in another, and changes automatically when the data changes.	User can create compound documents by embedding one document within another, or create reference buttons allowing quick navigation from one document to another.
Pointing	Emphasis is on precision. The mouse is precise, and reliance on visible targets requires precise targeting (for example, scrollbar buttons).	Emphasis is on casual interaction. Gestures can be made over larger regions (for example, flicking on a page to scroll).

Chapter 2 / Overview of the PenPoint User Interface

Basic Screen Layout: Notebook & Bookshelf

The basic PenPoint screen layout consists of a notebook for user data and a Bookshelf at the bottom of the screen for various system functions.

The figure below shows a typical notebook table of contents and Bookshelf:



▼ The Bookshelf

The Bookshelf contains the following icons by default:

- ? **Help.** Tapping the Help icon displays the Quick Help sheet. While the Quick Help sheet is displayed, the user can tap on any object or region of the screen to get a brief explanation of the object. Double-tapping on the Help icon opens the Help notebook, which contains more extensive help, organized by task for both the PenPoint™ operating system itself and for each installed application.
- ✓ **Settings.** The central place that the user goes to see and modify the settings for all the software in the machine. It is presented in the familiar form of a notebook, with sections for PenPoint User Preferences, Installed Software, and System Status.
-  **Connections.** A utility notebook that provides access to resources not physically located in the computer, including disks and printers connected either directly or via networks. Functions include importing files, installing applications, or connecting to network printers.
-  **Stationery.** A special notebook that contains stationery documents used as templates when the user creates new documents. When the user installs an application, PenPoint creates a section in the Stationery notebook with whatever stationery the application includes. The user can create custom stationery at any time by simply moving or copying an existing document into the Stationery notebook.
-  **Accessories.** For clocks, calculators, status gauges, and the like.
-  **Keyboard.** A simple accessory that the user can open at any time to simulate typing on a physical keyboard to enter characters directly into documents or input pads.
-  **In box.** Where documents that have been received via fax, email, file transfer, or other services show up.
-  **Out box.** When the user issues the Print command or Send command (regardless of whether the method is fax, email or some other service), PenPoint places the document in the Out box. When the computer can make the connection, PenPoint automatically sends the document.

The contents of the Bookshelf are not limited to the icons described above; the user can put ordinary documents on the Bookshelf as desired for quick access.

The main Bookshelf at the bottom of the screen is always present. In addition to the main Bookshelf, PenPoint also supports additional, disk-based bookshelves.

Each time a PenPoint disk is connected, either directly or over a network, a bookshelf appears showing the notebooks, documents, and accessories on that disk. The per-disk bookshelf appears and disappears automatically as the disk is connected and disconnected.

The additional bookshelves are normally obscured by the notebook. The user can expose the additional bookshelves either by closing the notebook or by dragging the bottom border of the notebook up to reveal the hidden bookshelves.

■ The Notebook

As a metaphor, the idea of the notebook is useful in many ways. The notebook concept is both simple and flexible, and capable of intuitively supporting a rich set of access mechanisms. The metaphor matches the device—notebooks are appropriate for small, take-anywhere devices.

✦ Paper as a Model for Presenting Information

We are all familiar with notebooks, forms, sheets, pads, tabs, bookmarks, sticky notes, and so forth. As far as possible, all the elements of the PenPoint user interface have been designed to make visual and conceptual sense in the context of this paper-like world.

Instead of control panels, data entry areas, and radio buttons, the user deals with option sheets, writing pads, and checklists. Instead of alerts for confirmation and error notification, PenPoint presents notes to the user. In fact, while the interface is implemented as an ordinary overlapping window system, the word “window” doesn’t even appear in the user documentation.

✦ The Notebook as a Model for Structuring Information

The basic user model for the PenPoint notebook is that each page of the notebook contains a single document.

The document itself may extend across many printed pages. The user can scroll the offscreen portions of the document into view by using the scroll margins or the flick gestures.

The user can choose to organize his or her notebook as a simple list of documents, or structure the documents into sections and subsections.

The user can create multiple notebooks. (For example, the user might create personal and work notebooks, or a notebook for each client.)

The information being viewed can be in memory or on secondary storage such as disk or CD-ROM.

✦ The Notebook as a Model for Accessing Information

The first page of the notebook is a table of contents showing all of the documents in the notebook.

The notebook metaphor allows the user to dispense with some of the conceptual baggage usually associated with computers. To work with a document, the user simply turns to the desired page in the notebook. There's no need to launch or quit applications, or to load or save files.

There are many ways to turn to pages, including tapping on the page number in the table of contents, flicking along the top of the page to flip to the next or previous page, and tapping a tab along the edge of the notebook.

The user can also access any page of the notebook by **floating** the page on top of the notebook. This is like temporarily unsnapping a page from a loose-leaf notebook. Double-tapping on a page number or tab floats the corresponding page.

✦ Gestures

The PenPoint operating system is unique in that it uses pen gestures as a fundamental means of interacting with the computer. Gestures are not just added on as equivalents to existing functions—the entire environment has been designed with gestures in mind.

The sections that follow describe two of the most important aspects of PenPoint's approach to gestures: the **dual command path** and **core gestures**.

✦ Dual Command Path

One of the basic principles of the PenPoint user interface is that the user should be able to issue any given command with either a visible control (such as a button or menu item) or a pen gesture. This is called the **dual command path**.

Each way of issuing commands has its advantage. Years of experience with GUIs has proven the value of putting commands on buttons and menus so the user can browse through them without having to remember the commands by name.

Gestures, on the other hand, have their own inherent advantages. They allow the user to easily specify both the operation and the operand in a single step. And of course they draw on the user's long experience in using a pen to write or draw.

Here are a few examples of how gestures and visible controls complement each other in the PenPoint operating system:

- ◆ Turning notebook pages. The user can either tap the arrows on either side of the page number at the right of the title line, or flick anywhere in the title line to turn to the next or previous page (flick left for next page, flick right for previous page).
- ◆ Scrolling. The user can either use the arrows or drag box on the scroll margin, or flick directly on the text or list to scroll it (flick up to shove the lines up, flick down to shove the lines down).
- ◆ Editing. Common editing operations such as **Delete**, **Edit**, and **Spell** are available from both the **Edit** menu and from gestures (cross-out for **Delete**, circle for **Edit**, S for **Spell**).
- ◆ Text formatting. Text styles such as **Bold**, **Italic**, **Underlined**, and **Normal** are all available from both the text option sheet and from gestures (the first letter of each command: B, I, U, and N).

Core Gestures

PenPoint defines a set of 11 core gestures whose meaning is consistent throughout the system. They are listed in the following table.

Core Gestures

Symbol	Name	Function
	Tap	Select
	Press	Move
	Tap-Press	Copy
	Flick	Scroll
	Cross Out	Delete
	Caret	Insert
	Circle	Edit
	Check	Options
	Undo	Undo

Core Gestures (Continued)

9

Pigtail

Delete

L

Down-Right

Insert Space

Consistent implementation of the core gestures is one of the keys to the transparency of the PenPoint user interface. The user quickly learns that throughout the system he or she can flick to scroll, circle to edit, check to get options, make a caret to insert, cross-out to delete, press to move, and tap-press to copy.

Chapter 3 describes the core gestures (as well as the other gestures supported by PenPoint) in detail.

■ Embedded Document Architecture (EDA)

All of the access mechanisms described so far—the table of contents, tabs, turning to pages, and floating pages—have their analogs in the physical world of notebooks and paper. But the PenPoint user interface goes beyond these direct corollaries. The power of the notebook metaphor is that it brings together the world of notebooks and paper with the world of the computer.

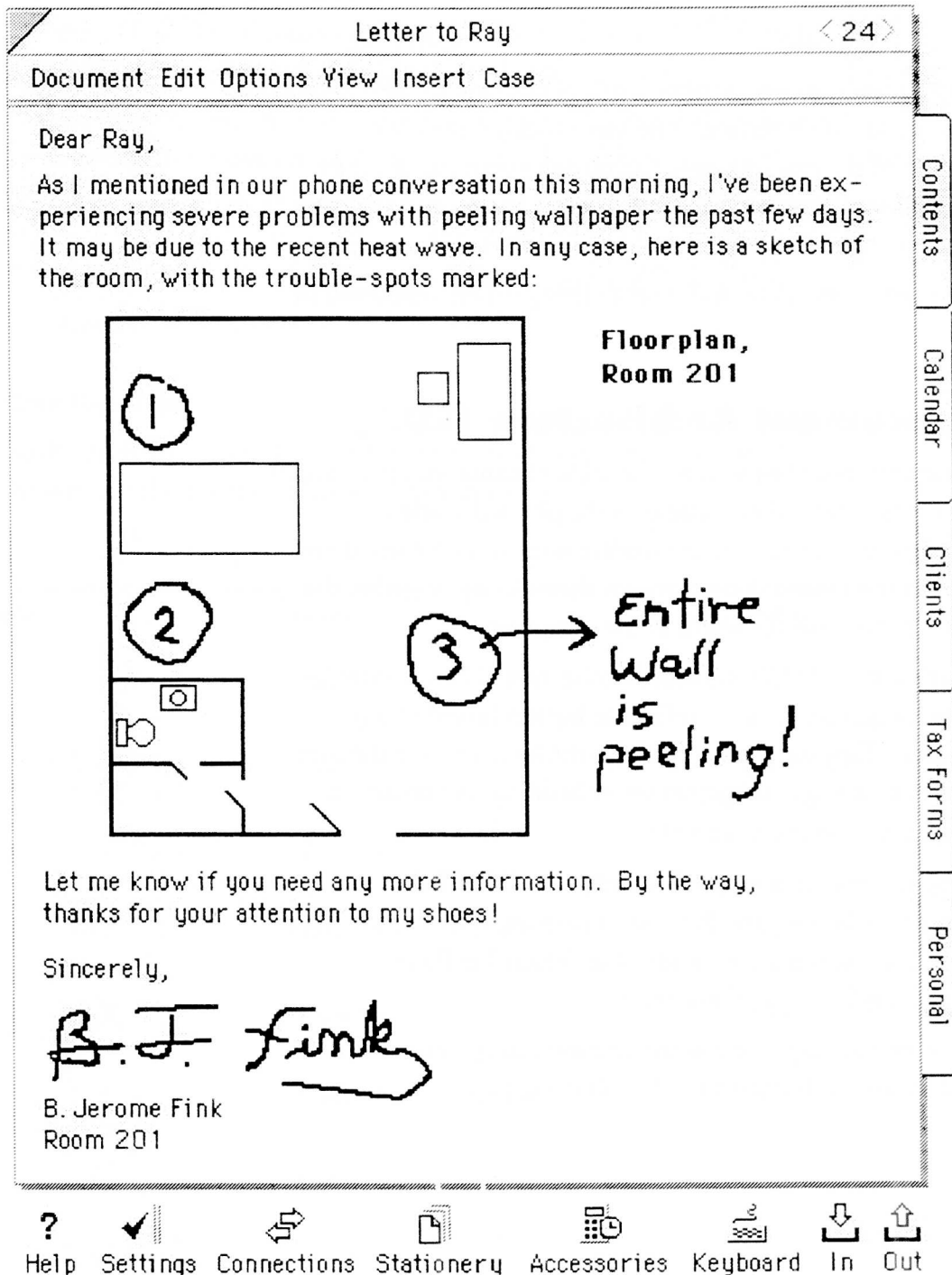
This is where PenPoint's Embedded Document Architecture (EDA) comes in. The user can, with a quick gesture, create a **reference button** linked to any location in any document. Tapping turns directly to the location. Or the user can eliminate the need to turn pages altogether by **embedding** one document within another to create a compound document.

Support for embedding and reference buttons is built into all PenPoint applications. This is possible because the PenPoint operating system is an object-oriented system. The EDA features are built into the default PenPoint application, and all other applications inherit them.

There are many uses for embedding. One of the common usage scenarios is to create compound documents, as described on the following pages.

Compound Documents

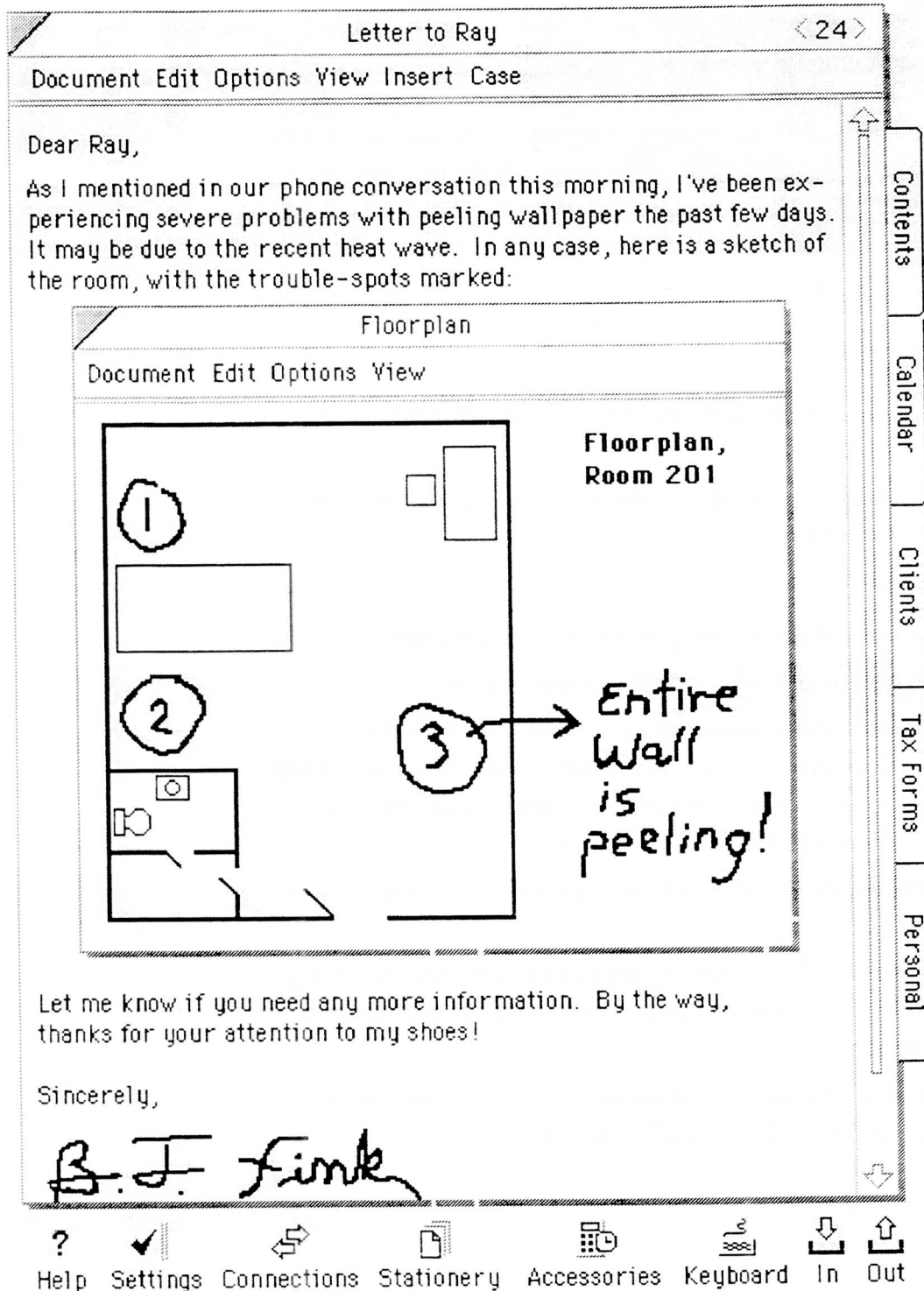
The next figure shows a text document containing an embedded drawing document:



The entire compound document is "live." The user can edit either the letter or the embedded drawing of the floor plan.

✦ Borders and Controls for Embedded Documents

The user can turn on the borders of the embedded document at any time to gain access to all the menus and controls:



Compound Document
(Borders On)

In keeping with the dual command path principle, there are two ways to toggle the borders of embedded documents on and off:

- ◆ By checking the appropriate option on the **Controls** option sheet.
- ◆ By making the standard “toggle borders” gesture (a short flick up followed by a flick down) anywhere in the body of the document.

Creating Embedded Documents

Creating an embedded document is as simple as dragging an existing document from the table of contents into the container document. Or the user can create a new embedded document by making the double-caret gesture at the desired location in the container document, which displays the **Create** menu showing the available Stationery documents.

Reference Buttons

Reference buttons allow the user to create customized access paths through his or her information.

The user can create reference buttons either to the document as a whole, or to a specified point within a document.

Common scenarios include:

- ◆ Buttons within one document for navigation to related documents.
- ◆ Buttons on the Bookshelf to mark a recently visited location.
- ◆ Buttons in a document’s cork margin for quick scrolling within the document. (The cork margin is an area at the bottom of each notebook page where the user can put reference buttons or embedded documents of any type. See “Basic Document Layout” in Chapter 6.)
- ◆ Buttons in a document that serve as an index into related documents, as in the Help notebook.

A reference button is a one-way link. Tapping the button once turns to the page in the notebook containing the document; double-tapping floats the document in front of the notebook.

To create a reference button, the user first selects the destination, then makes the double-circle gesture at the desired location of the button.

Part 2

User Interface

Components

THE PENPOINT USER INTERFACE

PART 2 / USER INTERFACE COMPONENTS

Chapter 3 / Gestures

23

Gesture Categories	23
Gesture Families	23
Core Gestures	24
Non-Core Gestures	25
Capital Letter Gestures	27

Pop-Up Edit Pads	47
Pop-Up Writing Pads	48
Embedded Writing Pads	49
Using Pads	50
Ruled/Boxed Pads	50
Notes	52

Chapter 4 / Controls

28

Buttons	28
Lists	29
Checklists	29
Pop-Up Checklists	30
Toggle Switch	30
Multiple Checklists	31
Checkboxes	32
Boxed Lists	32
Text Fields	32
Overwrite Fields	33
Fill-In Fields	33
Text Boxes	33
Gestures in Fields	34
Scroll Margins	35
Vertical Scroll Margins	35
Horizontal Scroll Margins	35
Scroll Margin Components	36
Scroll Margin Operations	36
Gestural Scroll Margins	37
Gauges	38
Subpage Controls	38
Palette Lines	39

Chapter 5 / Menus, Sheets, and Notes

40

Overview	40
Menus	41
Controls in Menus	41
Menu Behavior	42
Hierarchical Menus	42
Input Behavior for Sheets, Pads, and Notes	43
Multi-Page Sheets	43
Option Sheets	44
User Model	44
Relationship to the Selection	44
Response to Check Gesture	46
Clean and Dirty Controls	46
Writing and Edit Pads	47

Chapter 3 / Gestures

▼ Gesture Categories

The preceding chapter introduced the basic set of 11 core gestures. The PenPoint™ operating system actually supports many more gestures, some of which PenPoint defines for operations common across applications, and some of which PenPoint leaves open for application-specific uses.

The gestures fall into three categories:

- ♦ **Core Gestures.** The 11 fundamental gestures used throughout the system.
- ♦ **Non-Core Gestures.** PenPoint supports a larger set of gestures (approximately 30, depending on how they are counted) that are not universally applicable. Examples include variations on core gestures (such as multiple taps and multiple flicks), and other symbols (such as arrows and plus).
- ♦ **Capital Letter Accelerators.** PenPoint makes the set of capital letters available to the application developer to be used as gestures. The normal usage is to use the first letter of the command—for example, S for **S**pell or F for **F**ind. Application developers are free to assign commands to the capital letters as appropriate in the context of the application.

This chapter lists all of the core and non-core gestures, and all of the capital letter accelerators used by PenPoint itself. Other gesture lists appear in the appropriate chapter. For example, Chapter 4 lists the gestures accepted in text fields, Chapter 8 lists the gestures accepted by icons, Chapter 17 lists the gestures accepted by MiniText, and Chapter 18 lists those accepted by MiniNote.

▼ Gesture Families

One of the keys to the learnability and usability of PenPoint's gestures is the use of **gesture families**. A gesture family typically consists of a core gesture and several variations. By using similar shapes for similar meanings, gesture families help the user learn and remember gestures. For example, the caret family consists of the core caret gesture plus caret-tap and double-caret. All the caret gestures perform some type of insertion: Caret and caret-tap are for the primary and secondary interpretation of insert in any given context, and double-caret always displays the **Create** menu to insert an embedded document.

Below is a list of the gesture families for PenPoint 1.0.

- ♦ Taps: selecting objects and invoking controls.
- ♦ Flicks: bringing more information into view.
- ♦ Circles: editing.
- ♦ Carets: inserting and creating.
- ♦ Checks: options.
- ♦ Brackets: adjusting selected spans.
- ♦ Corners: (open meaning).
- ♦ Arrows: (open meaning).

▼ Core Gestures

Core Gestures



Tap. Selects an object or pushes a button.



Press. (Touch the pen to the screen and pause for a moment.) Initiates drag move when made over icons or selected application objects. Initiates drag area select when made over background or unselected objects in application.



Tap-Press. (Tap the screen once, then touch the pen to the screen again and pause for a moment.) Initiates drag copy.



Flick. (Four directions.) Flicking is used throughout PenPoint to bring more information into view. The user model is that the flick gesture shoves the object in the direction of the flick. Examples include scrolling text, turning notebook pages, and exposing overlapping tabs.



Cross Out. Deletes the most natural object in the context—a word in text, an entry in the Table of Contents, a figure in a graphics editor, and so forth.



Caret. Inserts or creates the most natural object in the context.



Circle. Edits.



Check. Displays the object's option sheet.



Undo. Equivalent to choosing Undo from the Edit menu.



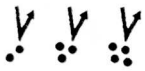
Pigtail. Deletes a single character in text, or a single character in boxed overwrite fields.



Down-Right. Inserts a single space in text. Inserts one or more spaces (depending on the length of the horizontal leg of the gesture) in boxed overwrite fields.

Non-Core Gestures

Non-Core Gestures



Double, Triple, and Quadruple Tap. In text, multiple taps are used to select successively larger units: a word, a sentence, or a paragraph. Double-tap is also used to float documents from icons, tabs, or reference buttons. Triple-tap is also used in several places to mean, roughly, “restore the default state.” For example, triple-tapping on a tab makes the tab’s label match the title of the document, triple-tapping on the title-line of an option sheet restores the sheet to its default size, and triple-tapping on a reference button links the button to the current selection and updates its label.



Brackets. Adjusts an existing selection.



Double, Triple, and Quadruple Flick. Available for application use to mean “scroll farther.”



Scratch Out. Deletes any object touched by the gesture.



Circle-Tap. This gesture is for the secondary meaning of edit in a particular context. For example, if circle edits one cell of a table, circle-tap might edit the entire line.



Circle-Line. Replace. In text, brings up an empty edit pad for the object under the gesture.



Double-Circle. Creates reference button linked to the current selection.



Circle-Flick-Down. In text, searches towards the end of the text for the next occurrence of the word (or selection) under the gesture.



Circle-Flick-Up. In text, searches towards the beginning of the text for the next occurrence of the word (or selection) under the gesture.



Caret-Tap. This gesture is for the secondary meaning of insert in a particular context. In text, for example, caret pops up an insertion pad, and caret-tap creates an embedded insertion pad.



Check-Tap. Available to applications to use for a secondary meaning related to options.



Up Arrow and Down Arrow. Whatever interpretation of “make the object larger/smaller” makes the most sense in the context.

Left-Arrow and Right Arrow. Available to applications to use for any meaning.

Double-Up Arrow and Double-Down Arrow. Available to applications to use as intensifiers for the meaning attached to up arrow and down arrow.

Double-Left Arrow and Double-Right Arrow. Available to applications to use as intensifiers for the meaning attached to left arrow and right arrow.

Non-Core Gestures (Continued)



Plus. Toggles selection. Used to toggle the selection on controls that use tap for invocation, such as icons and reference buttons. Appropriate in contexts where discrete, discontinuous selections make sense, such as graphics editors or lists.



Flick Left-Right. Toggles the mode of the application (if the application has modes.)



Flick Up-Down. Toggles borders.



Up-Right. In text, pops up a single-character pad for inserting a single character.



Up-Left. In MiniNote, left-aligns selected lines.



Down-Left. In text, inserts a paragraph break.



Right-Up. In text, capitalizes the first letter the word, or of each word in the selection. In MiniNote, closes white space.



Right-Down. In text, makes the word or selection lower case. In MiniNote, opens white space.



Left-Up. In MiniNote, closes white space.



Left-Down. In MiniNote, opens white space.



Down-Left-Flick. In text, inserts a line break.



Down-Right-Flick. In text, inserts a tab.



Right-Up-Flick. In text, capitalizes the word or selection.

Capital Letter Gestures

Capital Letter Gestures Used by PenPoint

Text

- B** Makes the word or selection bold.
- F** Brings up the Find sheet, set to start the search from the point of the gesture.
- I** Italicizes the word or selection.
- P** Proofs the word.
- N** Makes the word or selection “normal”—i.e. turns off bold, italic, and underlined attributes.
- S** Brings up Spell sheet, beginning checking from the point of the gesture.
- U** Underlines the word or selection.

Title Line

- C** Toggles cork margin on/off.
- M** Toggles menu line on/off.
- P** Prints document.
- T** Toggles tab on/off.

Miscellaneous

- T** In table of contents, toggles tab on/off.

Chapter 4 / Controls

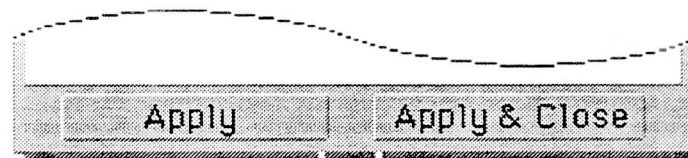
The PenPoint™ operating system provides a rich set of user interface components such as buttons, checkboxes, lists, and so on. Collectively, these components are referred to as **controls**.

This chapter describes the standard PenPoint controls. The application developer can also create customized controls.

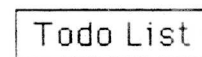
Buttons

The simplest type of control is the **button**, which allows the user to initiate an action.

There are several standard button forms, each tailored for a specific purpose. The figures that follow give examples of the most common forms of buttons:



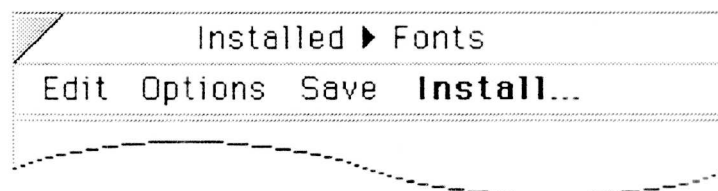
Raised Buttons on
Option Sheet



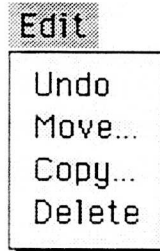
Outlined Reference
Button

- ☐ Lower Case Letters
- ☐ Upper Case Letters
- ☐ Numerals & Symbols
- ☐ Exit

Square Buttons with
Long Labels



Bold Button in Menu
Line



Unadorned Buttons in
Menu

▼ Lists

The PenPoint operating system provides several types of controls for the presentation of lists to the user.

▼ Checklists

Checklists allow the user to choose one item or setting from a list of choices. The choices can be either text or pictures, as shown in the examples below:

Orientation: ✓ Portrait Checklist with Text
Landscape Choices

Orientation: ✓



Checklist with Picture
Choices

✦ Pop-Up Checklists

Pop-up checklists present the choices in a more compact format. The currently selected choice appears after the list's label, separated by a small arrow. Tapping on the arrow or the current choice pops up a menu with the entire list:

Font: ▶ Roman

Font: ▶ Roman
Sans Serif

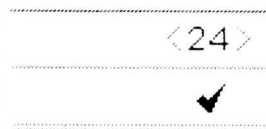
Pop-Up Checklist

The user can also use the flick and double-flick gestures to choose from a pop-up checklist in a single step, without bringing up the menu. A flick up or left makes the next item in the list current. Similarly, a flick down or right makes the previous item in the list current. A double-flick up or right makes the last item current, while a double-flick down or left makes the first item current.

✦ Toggle Switch

PenPoint provides a **toggle switch** as an alternative to checklists when there are only two choices. The control has two pictures, one of which is shown at any given time. Tapping toggles between the two.

A typical usage is to allow the user to switch between two input modes. In the example below, the pen signifies markup mode, the check signifies gesture mode:



Toggle Switches

Here's how the toggle switch works, step-by-step:

Toggle Switch Operation

- ✓ Initial state is gesture mode.
-  Touching the switch with the pen causes it to preview, showing the picture for markup mode.
-  Lifting the pen switches to markup mode.
-  Touching the switch again previews gesture mode.
- ✓ Lifting the pen switches to gesture mode.

Multiple Checklists

In some lists, the choices are not mutually exclusive—more than one choice can be on at a time.

For such situations, PenPoint provides a control called a **multiple checklist**, in which tapping on one choice toggles its check on and off without affecting the other choices in the list.

The figure below shows a typical multiple checklist:

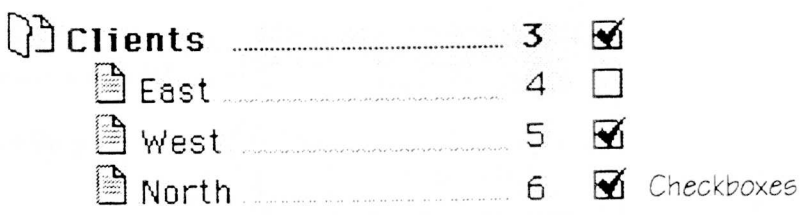
Controls: ✓ Menu Line
 ✓ Scroll Margin
 Cork Margin Multiple Checklist

The user can always tell whether a checklist is single or multiple by looking at the vertical line between the check and the list: A single line signifies a single-choice list, a double line signifies a multiple-choice list.

PenPoint uses this double-line convention to present multiple choice lists in dialog sheets, option sheets, and menus.

Checkboxes

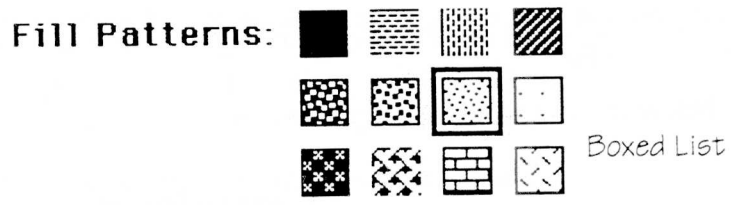
The PenPoint operating system also provides **checkboxes** for multiple choice lists. Checkboxes in the notebook table of contents provide an easy way for the user to toggle the tabs on and off for individual sections and documents:



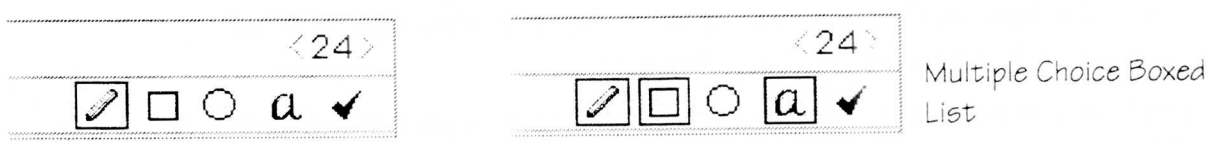
Boxed Lists

An alternative to checklists is the **boxed list**, in which the current choice is outlined instead of marked with a check.

The figure below shows a boxed list for choosing a fill pattern in a drawing program:



As with checklists, the choices in a boxed list need not be mutually exclusive. The figure below shows a boxed list from which the user can choose more than one translation mode:



Text Fields

PenPoint provides several types of fields for pen input:

- ◆ **Overwrite Fields.** Fields that display each character in a discrete box and allow direct overwrite of characters.
- ◆ **Fill-In Fields.** Single-line fields that are backed by pop-up editing pads
- ◆ **Text Boxes.** Boxes containing scrollable, fully editable text.

⚡ Overwrite Fields

For fields that are limited to a few characters, such as numeric fields, the PenPoint operating system provides a type of field that consists of a series of **overwrite boxes**, each of which accepts and displays a single character.

The figure below shows two overwrite fields, each with two boxes:

Time: : Overwrite Field

Overwrite fields are optimized for editing. To change a character, the user simply writes the new character directly over the existing one.

⚡ Fill-In Fields

Fill-in fields, in contrast to overwrite fields, are not boxed:

Name: Fill-In Field

When a fill-in field is empty, the user can write directly into it. After a short pause (as specified by the **Writing Timeout** preference on the Pen page of the Settings notebook), PenPoint translates the written characters and displays them in a standard computer font.

The user can also tap on the fill-in field to pop up an editing pad that presents the field contents in overwrite boxes, as described in the next chapter.

⚡ Text Boxes

The figure below shows a comments field implemented as a scrollable text box:

Comments

This report covers a boondoggle that Mr. Meyerowski took under the pretext of doing useful work for the company. We have photos of Mr. Meyerowski on a boat in somewhat

Text Box

Gestures in Fields

The following table describes the editing gestures accepted by overwrite and fill-in fields.

Gestures in Text Fields

Overwrite Fields



Press. Makes the box the focus of keyboard input.



Pigtail. Deletes the character in the box.



Down-Right. Inserts one or more spaces, depending on the length of the horizontal line.

Fill-In Fields



Tap. Observes the Preferred Input preference. In Pen mode (the default), pops up an edit pad containing the field's contents. In keyboard mode, sets the I-beam.



Double-Tap. Toggles selection of entire contents of field.



Press. If field contents are not selected, sets the I-beam. If field contents are selected, initiates move of field contents.



Tap-Press. If field contents are selected, initiates copy of field contents.



Cross Out. Deletes the contents of the field.



Scratch Out. Deletes any word touched by the gesture.



Circle. Pops up an edit pad containing the field's contents.



Caret. Pops up an empty writing pad targeted at the tip of the caret. The pad is ruled, boxed, or ruled/boxed, depending on the setting of the Pad Style preference.



Pigtail. Deletes the character under the pen-down point.



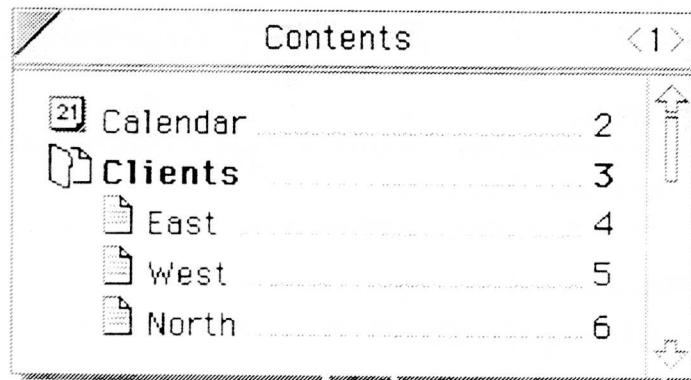
Down-Right. Inserts a single space at the pen-down point.

▼ Scroll Margins

In order to allow long documents or lists to be displayed in a smaller viewing region, PenPoint provides a control called a **scroll margin**.

▼ Vertical Scroll Margins

If there is more information past the top or bottom edges of the viewing region, then a vertical scroll margin appears on the right edge of the document:

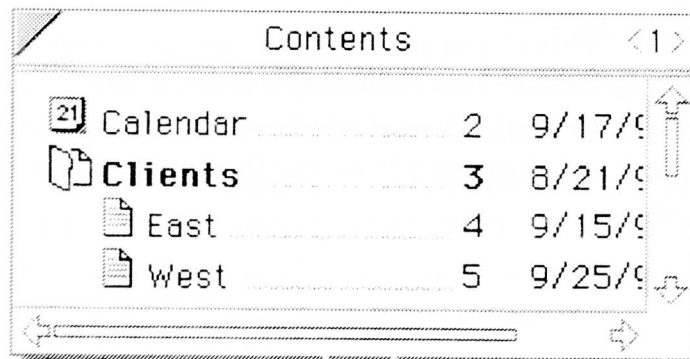


Vertical Scroll Margin

Left-handers can switch scroll margins to the left by setting the **Hand Preference** on the Fonts & Layout page in the Settings notebook.

▼ Horizontal Scroll Margins

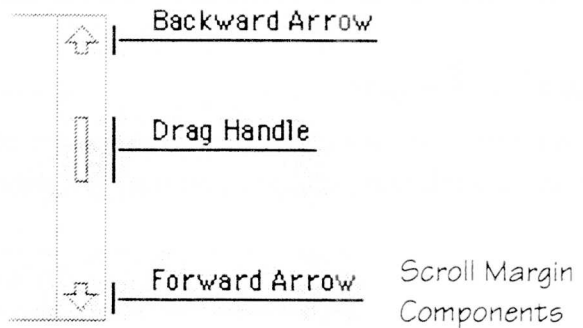
If there is more information past the left or right edge, a horizontal scroll margin appears at the bottom of the page. The figure below shows a document with both a vertical and a horizontal scroll margin:



Vertical and Horizontal Scroll Margins

✦ Scroll Margin Components

The figure below shows the components of a vertical scroll margin:



The size of the drag handle is inversely proportional to the size of the document: the longer the document, the smaller the handle.

✦ Scroll Margin Operations

Scroll margins support the following operations:

- ◆ **Scrolling by unit.** Tapping on either of the arrows scrolls by a single unit. In applications that are organized by lines, such as text or lists, the unit is a line. For tables, the unit is a row or column. For painting programs, the unit might be as small as a single pixel.
- ◆ **Scrolling by screenful.** A single tap above the handle scrolls towards the beginning of the sheet by one screenful. A tap below the handle scrolls towards the end of the sheet by one screenful.
- ◆ **Scrolling to beginning/end.** A double-tap above the handle scrolls all the way to the beginning of the sheet. A double-tap below the handle scrolls all the way to the end of the sheet.
- ◆ **Thumbing.** Dragging the handle of the scroll margin scrolls to the corresponding location in the sheet. Dragging the handle to the top scrolls to the beginning of the sheet. Dragging half way scrolls to the half-way point of the sheet, and so on.

⚡ Gestural Scroll Margins

The Fonts & Layout page of the Settings notebook has a control that allows the user to choose an alternate style of scroll margin. The alternate, Tap & Flick style, is a visually lightweight margin that has no visible controls:

Contents		<1>
21	Calendar	2
	Clients	3
	East	4
	West	5
	North	6

Tap & Flick Style of
Scroll Margin

The black portion of the margin's border is inversely proportional to the size of the document: the longer the document, the shorter the black line.

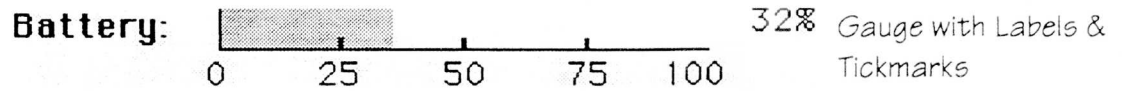
The Tap & Flick scroll margins support the following gestures:

- ◆ **Single Flick.** Sends the line under the start of the flick to the edge of the viewing region. Thus, flicking up on a line sends it to the top edge, and flicking down on a line sends it to the bottom edge.
- ◆ **Double Flick.** Scrolls as far as possible. Double-flicking up scrolls to the end of the sheet; double-flicking down scrolls to the beginning of the sheet.
- ◆ **Tap.** Scrolls to the corresponding location in the sheet. Tapping at the top scrolls to the beginning of the sheet. Tapping half way scrolls to the half-way point of the sheet, and so on.
- ◆ **Double-tap.** Scrolls to beginning or end. Double-tapping in the top half of the scroll margin scrolls all the way to the beginning of the sheet. Double-tapping in the bottom half scrolls all the way to the end of the sheet.

As always with flicking, the model PenPoint presents to the user is that he or she is manipulating a sheet of paper directly by shoving it with the pen. So flicking *up* moves the contents of the sheet *up*, and reveals information that had been hidden past the *bottom* edge. Likewise, flicking *down* moves the contents of the sheet *down*, and reveals information that had been hidden past the *top* edge.

▼ Gauges

The PenPoint operating system provides a gauge control for showing analog information graphically. The figure below shows a typical gauge:

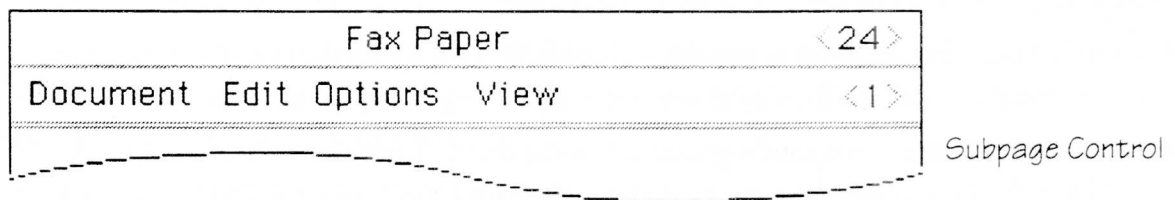


The application developer specifies whether gauges have tickmarks and labels, and also whether they are horizontal or vertical.

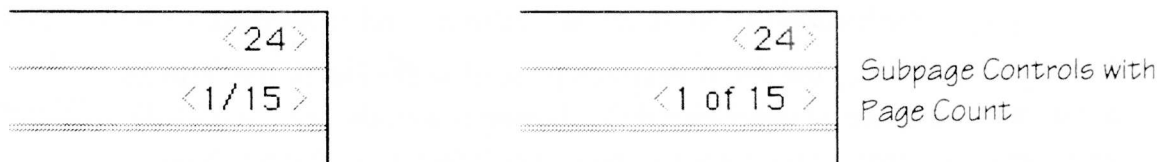
▼ Subpage Controls

To support applications that are organized into printed pages, PenPoint makes the page control that the notebook uses available as a standard toolkit control. Tapping on the left bracket turns to the previous page. Tapping on the right bracket turns to the next page. Pressing (as opposed to tapping) either of the brackets flips through pages.

The subpage control goes at the right of the menu line. In the example below, the Fax Paper document is on page 24 of the notebook, and page 1 of the Fax Paper document is in view:



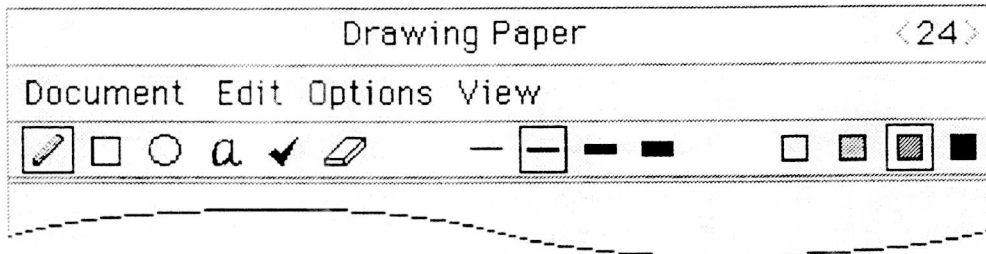
The Subpage control gives the application developer the option of showing the total number of subpages. Two formats are supported:



▼ Palette Lines

PenPoint applications often present their most frequently used controls in a palette line. Palette lines can be on any edge of the document's work area.

The figure below shows a typical palette line:



Palette Line with Boxed Lists

The palette line shown above has three boxed lists: for choosing input modes, pen widths, and ink colors.

Chapter 5 / Menus, Sheets, and Notes

Overview

The controls described in the previous chapter are typically presented in pop-up containers.

The PenPoint™ operating system provides four standard types of containers: menus, sheets, notes, and pads. The following table summarizes each with respect to usage, appearance, and input behavior.

PenPoint Containers

	Usage	Visual Conventions	Input Behavior
Menus	Used primarily for commands (such as buttons) but also for settings (such as checklists). Allow quick execution of a single command. Allow user to browse through the available commands.	Plain white sheet. No title. No Close Corner or Cancel button.	Modal—tapping outside the menu dismisses it. Tapping on a menu command takes down the menu and invokes the command. Tapping outside the menu dismisses it.
Dialog Sheets	Allow user to set parameters to commands (such as Find, Spell, Print).	White sheet on grey background. Centered title. Command buttons at bottom. Close Corner at upper left.	Usually modeless.
Option Sheets	For setting properties of objects or of the document as a whole. Option sheets are a specialized type of dialog sheet in which the command is “Apply the settings on the sheet to the selected object.”	White sheet on grey background. Centered title. Apply, Apply & Close buttons. Close Corner at upper left.	Modeless.
Writing Pads	For entering or editing text, labels, and fields. Used throughout the system.	White sheet on grey background. No title line. OK, Clear button. Cancel button at lower right.	Modal—tapping outside the pad flashes its borders.
Notes	Used for confirming operations that cannot be undone by the user and for displaying error messages.	White sheet on grey background. Left-justified, italic title. Buttons at bottom. Cancel button at lower right.	Modal—tapping outside the note flashes its borders.

The remainder of the chapter describes menus, sheets, pads, and notes in detail.

➤ Menus

⚡ Controls in Menus

Most menus consist of a simple list of buttons representing commands. However, menus can also contain checklists, multiple checklists, fill-in fields, and overwrite fields, as shown in the figures that follow:

Edit

Undo
Move
Copy
Delete

Menu with Commands

Show

☒ Icon
Size
Date
Tab Box

Sort

☒ By Name
☒ By Type
By Size
By Date

Menus with Single and Multiple Checklists

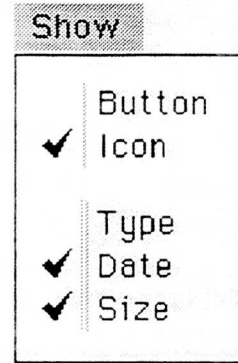
Alarm

☒ Daily
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
2/15/91

Size

☐	8	16
☒	10	18
	12	20
	14	7 2

Menus with Fill-In and Overwrite Fields



Single and Multiple
Checklists in a Single
Menu

⚡ Menu Behavior

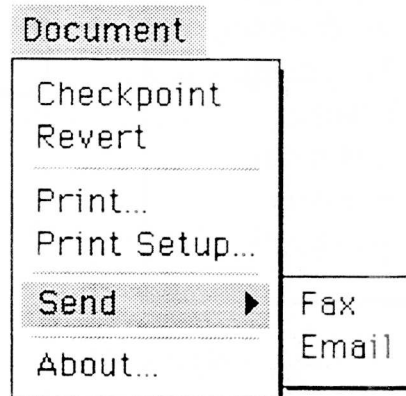
The user taps on the menu's label to display the menu, then taps on a control in the menu to take the desired action and dismiss the menu. While a menu is displayed, tapping anywhere outside the menu dismisses it.

There are two exceptions to this basic behavior:

- ◆ All other menu labels on the screen remain active. This allows the user to browse through a group of menus by tapping each label without needing to dismiss the previously opened menu each time.
- ◆ The menu doesn't close after the user writes or gestures into a text field (fill-in or overwrite) within the menu. This allows the user to verify that the contents of the field are correct before dismissing the menu.

⚡ Hierarchical Menus

A right-pointing arrow to the right of a menu command tells the user that tapping that command will display a submenu:



Menu and Submenu

Input Behavior for Sheets, Pads, and Notes

There are two basic models for how sheets, pads, and notes handle pen input:

- ♦ **Modal.** Modal objects grab all pen input. Any time a modal pad or note is on the screen, the user must dismiss it before interacting with the rest of the system. Tapping elsewhere on the screen flashes the borders to draw the user's attention to the modal object.
- ♦ **Non-Modal.** Non-modal sheets don't interfere with input to other areas of the screen. This allows the user to leave them up and use them as needed.

This important distinction is reflected in a visual convention:

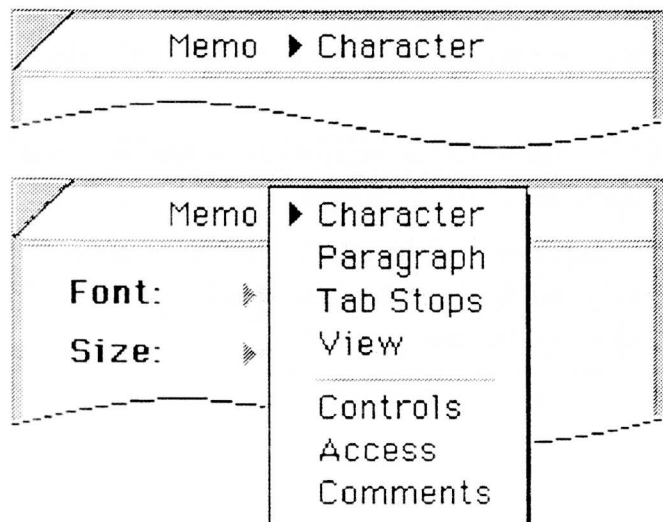
- ♦ All modeless objects have a **Close Corner** at the left of their title lines. This includes option sheets, dialog sheets, notebooks, and floating documents.
- ♦ All modal objects have a **Cancel** button at the right of their command lines. This includes writing and editing pads, confirmation notes, error notes, and special sheets such as the Quick Help sheet.

This convention allows the user to tell at a glance whether something is modeless or modal.

Multi-Page Sheets

Both dialog and option sheets can have multiple pages for different categories of options.

Sheets with multiple pages have a pop-up list in the title line allowing the user to turn to any page. Tapping on the arrow displays the menu with the names of all the pages:



Pop-Up Menu for
MiniText Option Sheet

➤ Option Sheets

Option sheets are used widely throughout the PenPoint operating system to allow the user to modify:

- ◆ Document-wide options. Standard document options implemented by the PenPoint Application Framework include the title and author of the document, whether the document is editable and deletable, whether its menu line and cork margin are shown, and so on. Applications may add other document-wide options, such as whether or not the background grid is shown in a drawing program or whether a spreadsheet is automatically recalculated each time a value is changed.
- ◆ Objects within documents (such as figures in a drawing document or text in a word processor). Typical text options include font, point size, and paragraph justification.

➤ User Model

The user model is that each document has a single, multi-page option sheet.

There is a standard **Options** menu whose entries are the titles of the option sheet pages. The **Options** menu shows the three default document option sheets—**Comments**, **Access**, and **Controls**. Each application adds option sheet pages as appropriate.

The user brings up an option sheet either from the **Options** menu or by making the check gesture over the object of interest.

➤ Relationship to the Selection

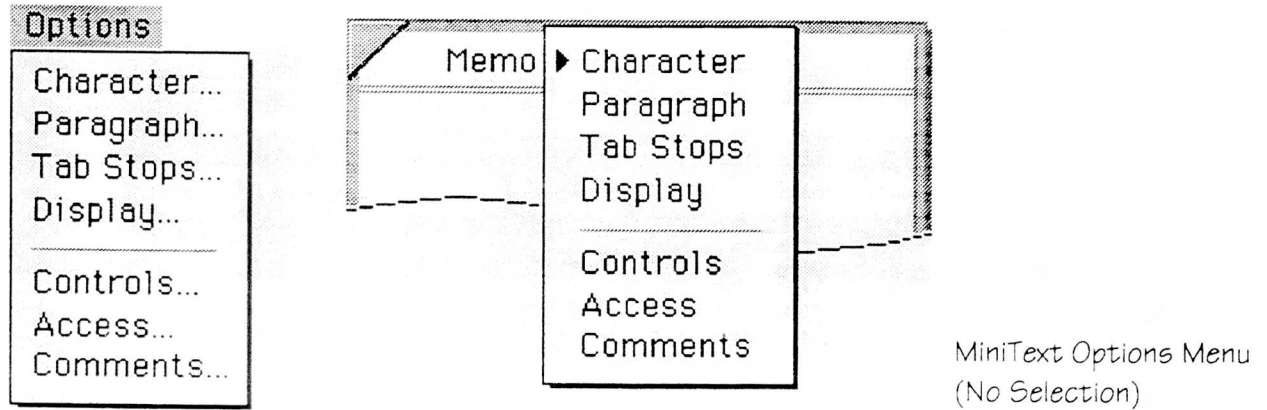
The available pages (and the corresponding menu entries) change as the selection changes, so they always show the options applicable to the currently selected object.

The application developer specifies which of the application-specific sheets are applicable when nothing is selected in the document.

Any time an option sheet is displayed and the current selection is not of the option sheet's type, the **Apply** and **Apply & Close** buttons become inactive because there is no target to apply the settings to.

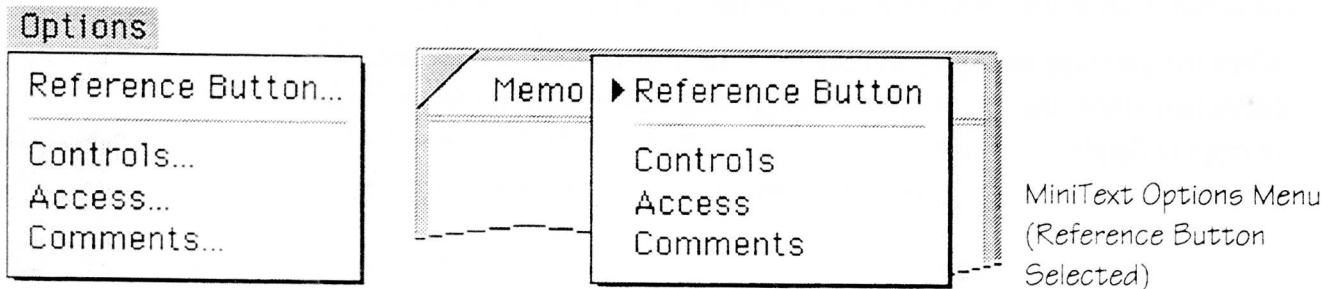
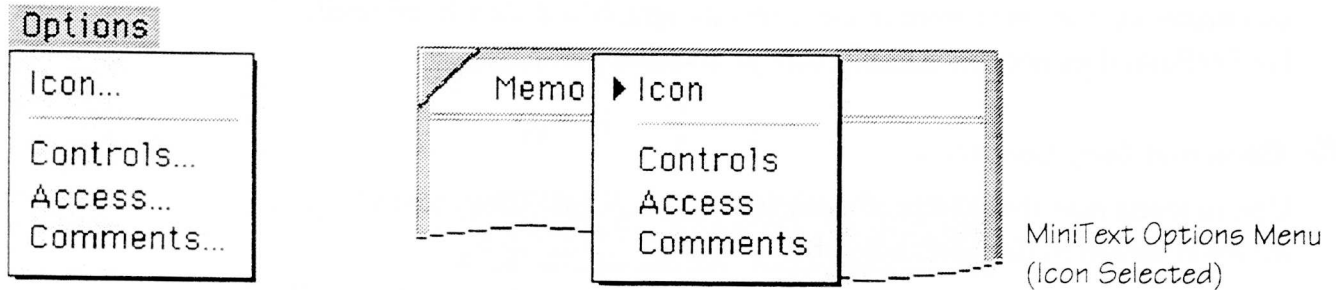
The figures that follow show how the **Options** menu for the MiniText application changes with the selection.

The figure below shows the menu when nothing in the document is selected:



The text-related pages appear first, separated by a line from the standard pages that apply to the entire document.

When the user selects something within the document, the names of the pages applicable to the selected object appear in the **Options** menu. The next two figures show the **Options** menu when the selection is an embedded document icon or a reference button:



⚡ Response to Check Gesture

When the user draws the check gesture over an object that is not selected, the PenPoint operating system first selects the object that was the target of the gesture, then displays its option sheet.

PenPoint selects the most natural object, depending on the context. For example, in the table of contents the default object is the document, while in text the default object is the word (not the character) under the gesture.

If an option sheet is already up when the user draws a check on another object (of the type appropriate to the top-most page of the option sheet), PenPoint updates the sheet's contents to reflect the current state of the object under the gesture. If the object is of a different type than the top-most page, PenPoint displays the appropriate page.

Note that there is not necessarily a one-to-one relationship between option sheet pages and types of objects; a given page can be applicable to more than one type of object, and vice versa. The application developer specifies which pages are applicable to which types of objects.

For example, in MiniText, drawing a check over a word when the option sheet is not displayed brings up the **Character** page. But the **Paragraph** page is also applicable when a word is selected. So if the **Paragraph** page is displayed, and the user makes a check over a word in a different paragraph, the sheet is refreshed, but PenPoint does not automatically turn to the **Character** page.

⚡ Clean and Dirty Controls

Option sheets have the concept of **clean** and **dirty** controls. Clean controls are shown in dark grey; dirty controls in black.

When the sheet initially comes up (or is updated by the drawing of a check), all the controls are clean. When the user modifies a control, it is marked dirty.

When the user taps the **Apply** button, only the dirty controls are applied. This distinction allows the user to tell unambiguously which options will be affected by tapping **Apply**.

Let's take a common example from text. Suppose an entire paragraph is selected. Most of the font is normal, but there are a few bold words and a few italic words. Since the setting would take its value from the first letter, neither bold nor italic would be checked. The user needs to know whether the entire paragraph will be made not bold and not italic when **Apply** is tapped. The clean/dirty convention satisfies this concern.

If the user selects another object while an option sheet is displayed, the toolkit automatically marks all the controls as dirty.

This allows the user to easily copy properties from one object to another by simply bringing up the option sheet for the source object, selecting the destination object, and tapping **Apply**.

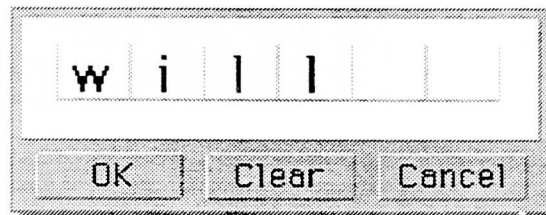
Double-tapping on the title line of the option sheet toggles the clean/dirty state of all the controls.

✦ Writing and Edit Pads

This section describes the pads for entering and editing text.

✦ Pop-Up Edit Pads

In PenPoint, the circle gesture brings up an **edit pad** for the object under the circle. Edit pads are built around overwrite fields:



Pop-Up Edit Pad

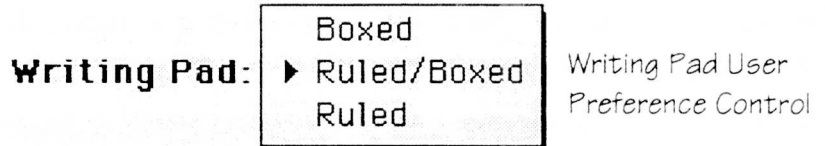
Edit pads are used extensively in PenPoint to provide a uniform way for the user to edit fields and labels of all sorts, including:

- ◆ Titles on the document title line.
- ◆ Document and section names in the table of contents.
- ◆ File names in the Connected Disks page of the Connections notebook.
- ◆ Icon labels.
- ◆ Tab labels.
- ◆ Words and phrases in MiniText and MiniNote.

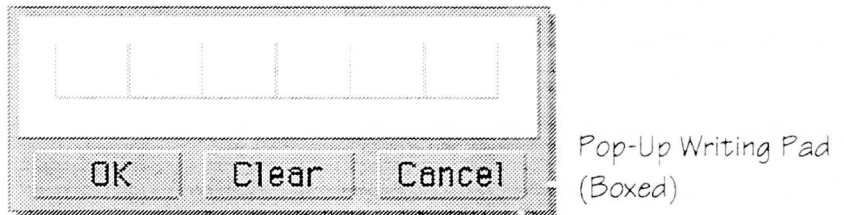
✦ Pop-Up Writing Pads

In MiniText, the caret gesture pops up a writing pad to allow the user to insert new text.

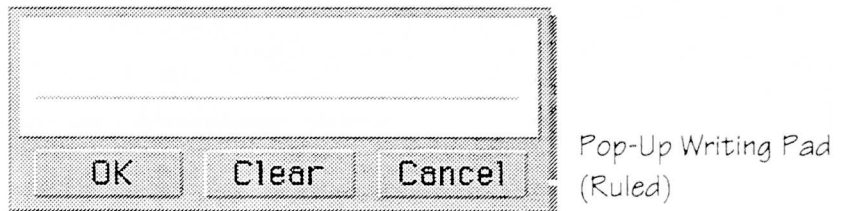
The Writing page of the Settings notebook has a control for the pad style:



The Boxed style is simply an empty edit pad:



The Ruled style is unsegmented:



✦ Embedded Writing Pads

In MiniText, the caret-tap gesture creates a writing pad that is embedded in the text stream, instead of overlapping the text as do the pop-up pads.

Like pop-up writing pads, embedded writing pads also observe the user preference for pad style. The figures below show ruled and boxed embedded writing pads:

A diagram of a ruled embedded writing pad. It features a rectangular frame with a dashed border. At the top, there are three buttons labeled "OK", "Clear", and "Dismiss". Below the buttons, the writing area is filled with horizontal lines, resembling ruled paper. The entire pad is shown as if it is embedded within a larger text stream, indicated by a dashed line passing through the bottom of the pad.

Embedded Writing Pad
(Ruled)

A diagram of a boxed embedded writing pad. It features a rectangular frame with a dashed border. At the top, there are three buttons labeled "OK", "Clear", and "Dismiss". Below the buttons, the writing area is filled with a grid of small squares, resembling a boxed or ledger-style writing pad. The entire pad is shown as if it is embedded within a larger text stream, indicated by a dashed line passing through the bottom of the pad.

Embedded Writing Pad
(Boxed)

Note that there are visual differences between pop-up and embedded pads.

While the pop-up pads have shadows and a grey background to make them stand out, the embedded pads have no shadow, and a lightweight, paper-like look.

Also, the right-most button reads Cancel on the pop-up pads and Dismiss on the embedded pads. This reflects the PenPoint convention that sheets have a Cancel button if (and only if) they are modal.

Pop-up and embedded pads do not differ with respect to their usage and behavior.

Using Pads

Boxed pads (both edit pads and boxed writing pads) have two different modes of behavior with respect to translation:

- ◆ Strokes entered into blank pads, or to the right of the last occupied box in a pad that contains already-translated characters, remain as written without being translated automatically after a timeout. Translation occurs when the user taps OK.
- ◆ Strokes entered into a box that already contains a translated character (or into a blank box that has a box with a translated character to the right of it) are translated automatically when the user lifts the pen from the screen for more than a moment. The timeout is user-settable, and defaults to .6 of a second.

This behavior is designed to support the most common usage scenario, in which the user first enters the word or phrase into a blank pad, then taps OK to translate, then overwrites the characters as needed to correct any translation errors.

The buttons on boxed pads work as follows:

- ◆ **OK.** If there are any untranslated strokes, tapping OK translates the contents of all the boxes. If there are no untranslated strokes, then tapping OK returns the contents of the pad to the caller and dismisses the pad.
- ◆ **Clear.** Clears the pad.
- ◆ **Cancel.** Dismisses the pad without making any change to the object being edited.

Ruled pads behave differently from both boxed pads and fill-in fields. Ink in ruled pads is *not* translated automatically after the handwriting timeout has elapsed. The untranslated strokes remain in the pad until the user taps OK, at which point the strokes are translated and the resulting text string is inserted into the text stream.

Ruled/Boxed Pads

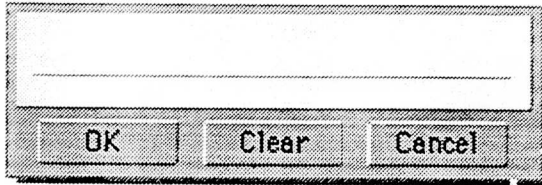
The **Ruled/Boxed** style of writing pad combines the best features of ruled and boxed pads: the naturalness of writing into a non-segmented area, and the convenience of correcting characters by overwriting.

The following example describes the behavior of ruled/boxed pads in detail:

Using Ruled/Boxed Writing Pads

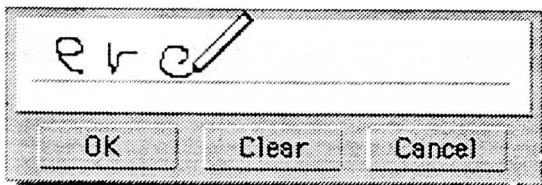
Able was I saw Elba.

User makes caret gesture, with tip of caret targeted where new text is to be inserted.



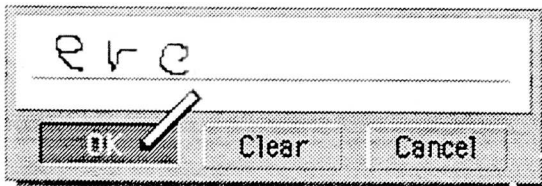
Blank ruled pad pops up.

Able was I saw Elba.



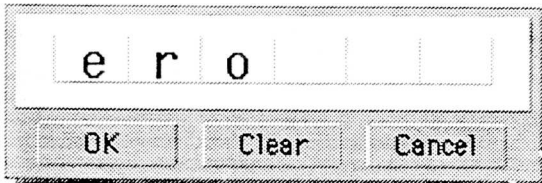
User writes in pad.

Able was I saw Elba.



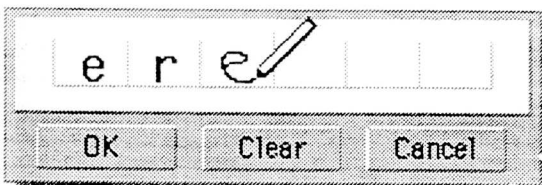
User taps OK to translate.

Able was I saw Elba.



The translated text is presented in overwrite boxes for easy correction.

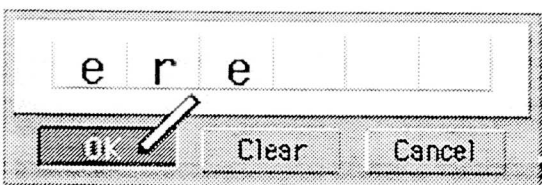
Able was I saw Elba.



User corrects characters as needed by overwriting.

Translation is triggered by a timeout, as specified by the Writing Timeout preference.

Able was I saw Elba.



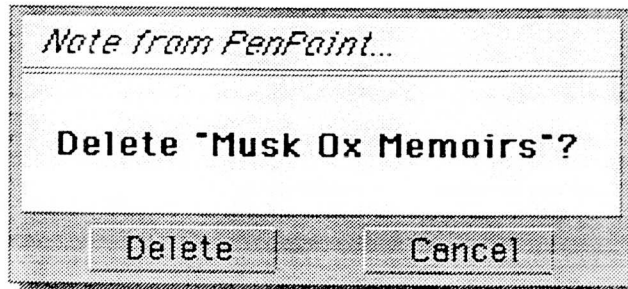
When all corrections have been made, user taps OK to insert contents of pad into the text stream

Able was I saw Elba.

Notes

PenPoint applications use modal notes to inform the user of errors and to ask the user to confirm operations that can't be undone.

The figure below shows a typical confirmation note:



Confirmation Note for
Delete

Part 3

Standard User

Interface Features

THE PENPOINT USER INTERFACE

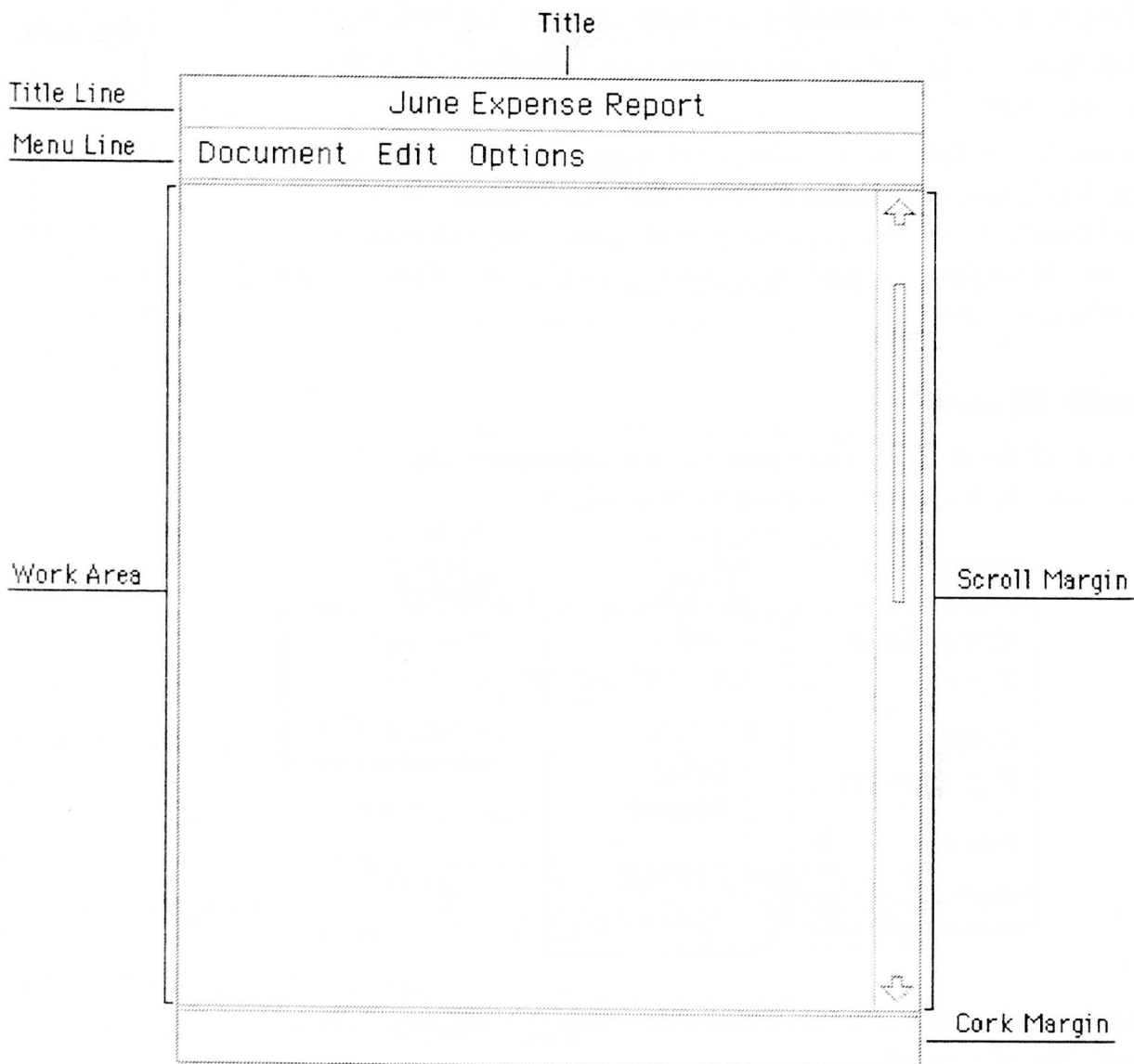
PART 3 / STANDARD USER INTERFACE FEATURES

▼ Chapter 6 / Standard Menus, Option Sheets, and Commands	55
Basic Document Layout	55
Default Menus	56
Document Menu	57
Edit Menu	62
Options Menu	66
▼ Chapter 7 / Drag & Drop Interface	68
User Model	68
Drag & Drop Operation	68
Example: Moving a Word in Text	69
Import and Export	69
Name-Clash Notes	71
▼ Chapter 8 / Icons	73
Icon Usage	73
Response to Gestures	74
Open and Closed Appearance	74
Icon Sizes	74
Icon Option Sheet	75
Standard PenPoint Icons	76
▼ Chapter 9 / Reference Buttons	79
Creating Reference Buttons	79
Response to Gestures	79
Reference Button Option Sheet	80
▼ Chapter 10 / Busy Clock	81

Chapter 6 / Standard Menus, Option Sheets, and Commands

Basic Document Layout

The following figure shows the basic components and layout for documents:



Basic Document Layout

The standard document components are:

- ◆ **Title Line.** Contains the title of the document.
- ◆ **Menu Line.** Each label in the menu line represents a category of commands.
- ◆ **Scroll Margin.** While each document occupies only one page of the notebook, the document itself may be of any length. The scroll margin allows the user to bring offscreen portions of the document into view.
- ◆ **Cork Margin.** A margin below the body of the document into which the user can place icons (useful for pop-up notes or annotations) and reference buttons (useful for scrolling directly to specific places in the document or for linking to other documents). The cork margin is not shown by default.
- ◆ **Work Area.** The bulk of the page is reserved for the information in the document itself.

The user can show or hide the menu line, scroll margin, and cork margin at any time from the Controls option sheet (described later in this chapter under “Options Menu”). Or the user can accomplish the same thing with gestures on the title line: M toggles the menu line, double-flick toggles the scroll margin, and C toggles the cork margin.

▼ Default Menus

By default, PenPoint documents have three menus: Document, Edit, and Options. The default contents of the menus are as follows:

Document	Edit	Options
Checkpoint	Undo	Controls...
Revert	Select All	Access...
Print...	Move...	Comments...
Print Setup...	Copy...	
Send ▶	Delete	
About...	Find...	
	Spell...	

Contents of Default
Menus

Application developers tailor the contents of these menus, and add other menus to the menu line, as appropriate.

Document Menu

The **Document** menu has the following commands by default:

Checkpoint. Files the document in its current state.

Revert. Restores the document to the state it was in at the last checkpoint.

Print. Displays the **Print** sheet for printing the document..

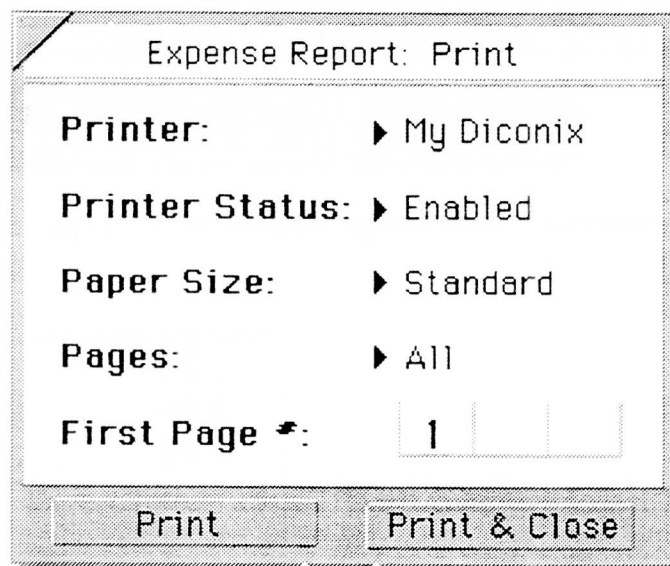
Print Setup. Displays the **Print Setup** sheet for setting print-related options such as headers and footers, margins, and so on.

Send. Displays a submenu with commands to invoke each of the currently installed services, such as fax and email.

About. Displays the About sheets: **About Document** and **About Application**.

Print

The **Print** sheet contains the settings that are those most closely related to the print job itself, and therefore most likely to be changed by the user at the time of printing:

A screenshot of a dialog box titled "Expense Report: Print". The dialog box has a light gray background and a dark gray border. It contains five rows of settings, each with a label on the left and a value on the right. The labels are "Printer:", "Printer Status:", "Paper Size:", "Pages:", and "First Page #:". The values are "My Diconix", "Enabled", "Standard", "All", and "1" respectively. The "First Page #" value is entered in a text field. At the bottom of the dialog box, there are two buttons: "Print" and "Print & Close".

Expense Report: Print	
Printer:	My Diconix
Printer Status:	Enabled
Paper Size:	Standard
Pages:	All
First Page #:	1
Print Print & Close	

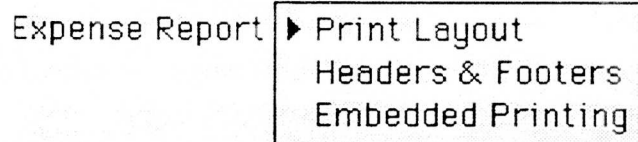
Print Sheet

The list of printers is taken from the **Connected Printers** page of the **Connections** notebook (described in a later section).

Print Layout

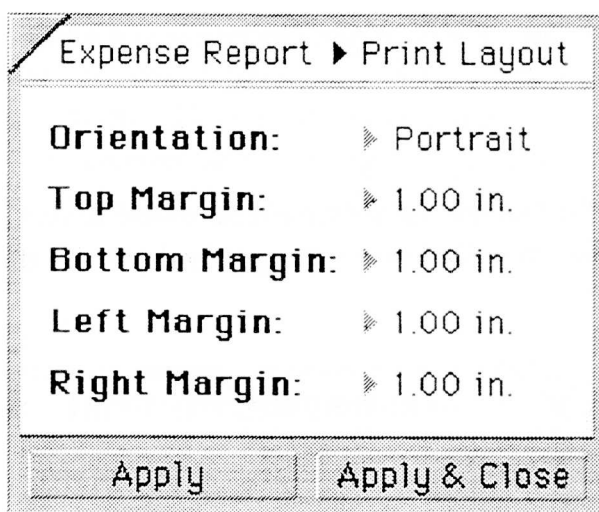
The three **Print Setup** sheets are for parameters that typically remain relatively constant across print jobs, and therefore are most usefully thought of as properties of the document.

A pop-up menu on the title line allows the user to access the desired sheet:



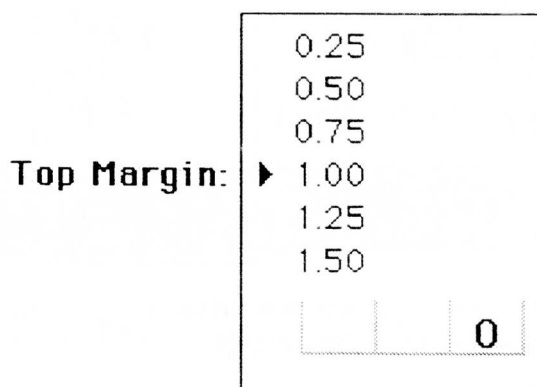
Pop-Up Menu for Print Setup Sheets

The figure below shows the **Print Layout** sheet:



Print Layout Sheet

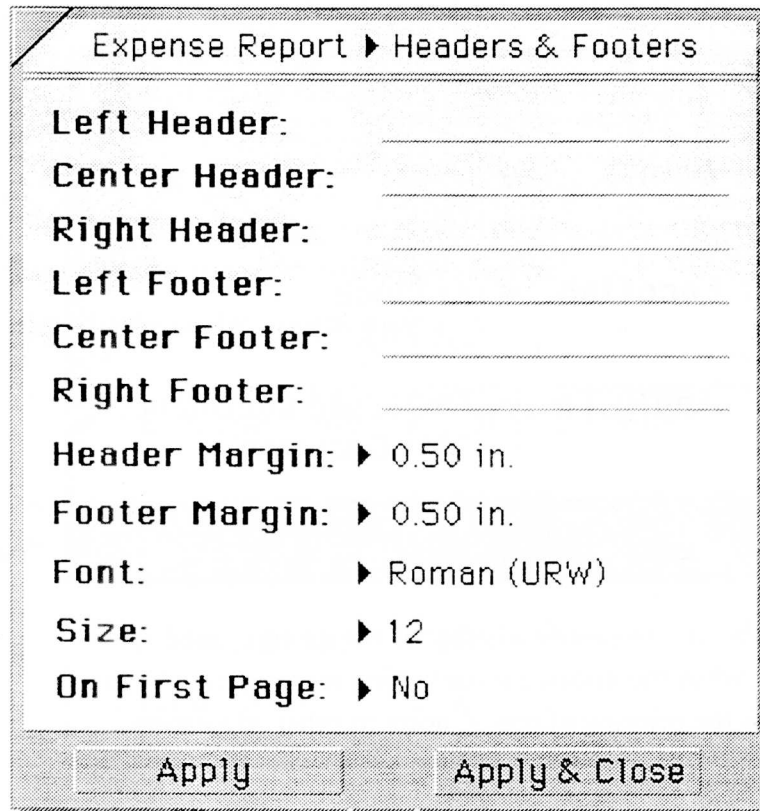
The pop-up menu for the margins contains a range of common settings plus an overwrite field:



Pop-Up Menu for Margins

⚡ Headers & Footers

The figure below shows the Headers & Footers sheet:



Expense Report ▶ Headers & Footers

Left Header: _____

Center Header: _____

Right Header: _____

Left Footer: _____

Center Footer: _____

Right Footer: _____

Header Margin: ▶ 0.50 in.

Footer Margin: ▶ 0.50 in.

Font: ▶ Roman (UR'w)

Size: ▶ 12

On First Page: ▶ No

Apply Apply & Close

Headers & Footers
Sheet

✦ Embedded Printing

The **Embedded Printing** sheet controls the printing of the document when it is embedded:

Expense Report > Embedded Printing

Print: ☒ Yes
☐ No

Location: ☒ In Place
☐ At End

Apply To: ☒ Embedded Documents
☐ This Document

Apply Apply & Close

Embedded Printing
Sheet

The **Print** control allows the user to specify whether or not an embedded document is to be printed when the document containing it is printed. A typical usage would be to suppress the printing of posted notes or other annotations within a document.

The **Location** control determines where the embedded document is printed in relation to the document containing it. **At End** causes the embedded document to be extracted from its place and printed, in its entirety, following the end of the container document. **In Place** causes the embeddee to be printed basically as it appears on the screen. Note that the entire contents of the embedded document are not necessarily visible; any part of the document that is scrolled off screen will not be printed.

When printing embedded documents **In Place**, the PenPoint™ operating system takes the **Print Layout** and **Headers & Footers** settings from the container document, not the embeddee. PenPoint does one other bit of special handling for **In Place** documents by printing them as if **Show** on the **Controls** sheet were set to **Borders Only**, so the title line, menu line, scroll margin, shadow, and resize handles are never printed. PenPoint prints the border as specified by the **Borders** control: **None**, **Single**, **Double**, or **Dashed**.

The **Embedded Printing** sheet appears on top-level documents as well as embeddees. Setting the **Apply To** control to **Embedded Documents** is a shortcut for going to all the embedded documents and setting their options individually.

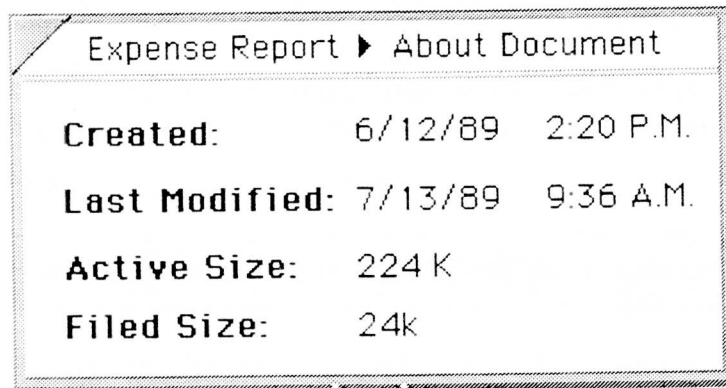
Typically, the user will set the options at the top level—for example, to suppress printing of embedded notes within a document, or to print all the embedded documents at the end as a kind of appendix.

The user can also set the options for individual embeddees. For example, the user can specify that embeddees are printed at the top level, and suppress the printing of one particular embeddee by turning printing off on its option sheet.

⚡ About Document and About Application

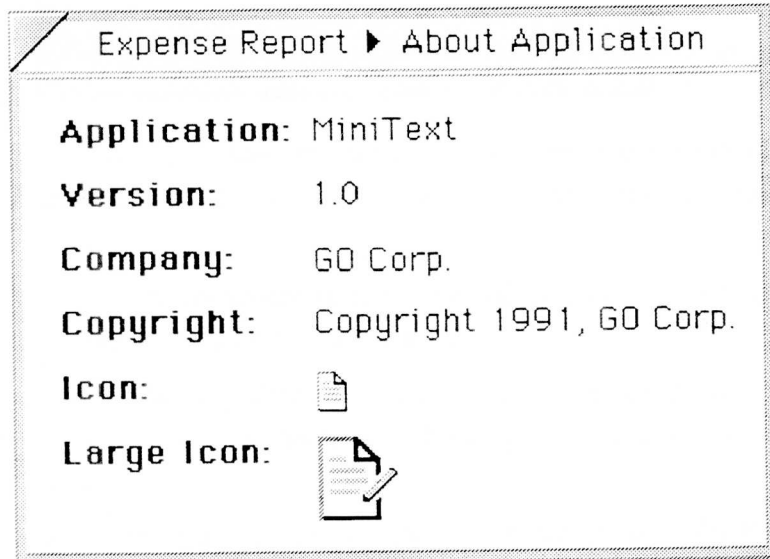
The **About** command on the **Document** menu gives access to read-only information about the document. There are two sheets: one for the document itself and one for the application associated with the document.

The figure below shows the **About Document** sheet:



About Document Sheet

The figure below shows the **About Application** sheet. The developer of the application typically customizes this sheet to show information such as the company's logo, the author of the application, and so on.



About Application Sheet

Edit Menu

The Edit menu has the following commands by default:

Undo. Reverses the effect of the most recently executed operation.

Select All. Selects the entire contents of the document.

Move. Puts the selected object into **Move** mode (see Chapter 7 for details).

Copy. Puts the selected object into **Copy** mode (see Chapter 7 for details).

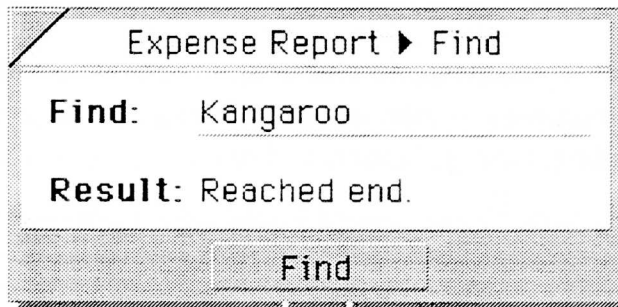
Delete. Deletes the selected object.

Find. Displays the **Find** sheet, with the starting point for the search set to the beginning of the document.

Spell. Displays the **Spell** sheet, with the starting point for the spelling check set to the beginning of the document.

Find

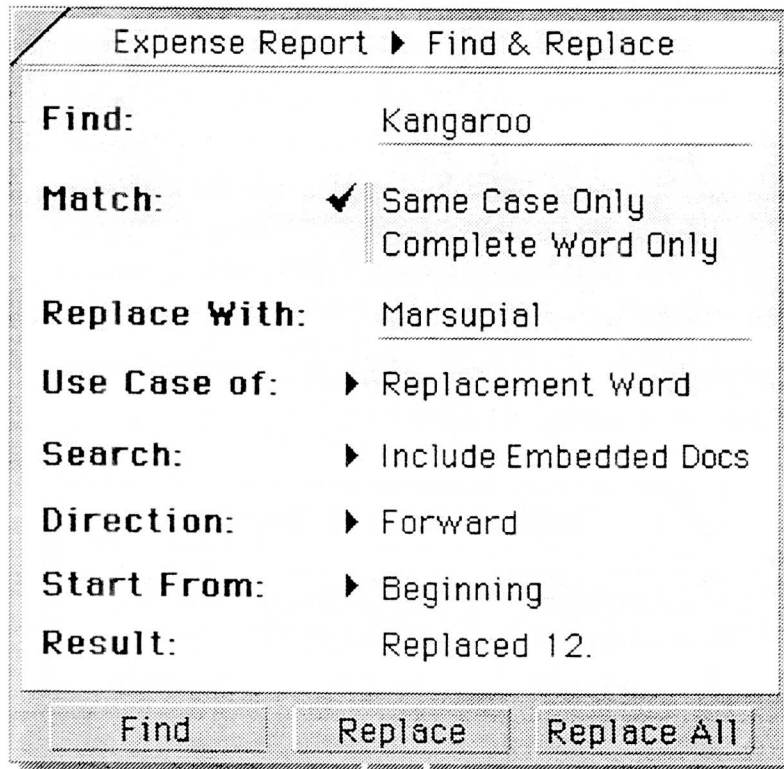
The default **Find** sheet has a fill-in field for the target and a read-only field for the result of the most recent search:

A screenshot of a 'Find' dialog box titled 'Expense Report ► Find'. It contains two text fields: 'Find:' with the text 'Kangaroo' and 'Result:' with the text 'Reached end.'. Below these fields is a button labeled 'Find'.

Expense Report ► Find	
Find:	Kangaroo
Result:	Reached end.
Find	

Find Sheet

The Find & Replace sheet allows the user to specify various parameters affecting both the find and the replace operations:



The image shows a dialog box titled "Expense Report Find & Replace". It contains several labeled fields and options:

- Find:** A text field containing the word "Kangaroo".
- Match:** A section with two radio button options: "Same Case Only" (which is selected with a checkmark) and "Complete Word Only".
- Replace With:** A text field containing the word "Marsupial".
- Use Case of:** A dropdown menu currently showing "Replacement Word".
- Search:** A dropdown menu currently showing "Include Embedded Docs".
- Direction:** A dropdown menu currently showing "Forward".
- Start From:** A dropdown menu currently showing "Beginning".
- Result:** A text field showing "Replaced 12.".

At the bottom of the dialog box are three buttons: "Find", "Replace", and "Replace All".

Find & Replace Sheet

The user brings up the sheets either by choosing **Find** from the **Edit** menu or by making the **F** gesture. Which of the two sheets is shown—**Find** or **Find & Replace**—depends on which sheet the user has last turned to. Regardless of which sheet is shown, the search observes the settings of the controls on the **Find & Replace** sheet.

Choosing **Find** from the menu or making an **F** gesture over the title line brings up the sheet with **Start From** set to either **Beginning** or **End**—depending on the setting of **Direction**.

If **Start From** is set to **Beginning** or **End**, the search will start from the beginning or end of the immediate document (for example, the embedded document, not the container).

Making an **F** gesture within the document selects the word under the gesture and brings up the sheet with **Start From** set to **Last Match**.

If **Search** is set to **Include Embedded Docs**, then the search will descend into any embedded documents that are open (in place—not floating) and back out again to the outermost document.

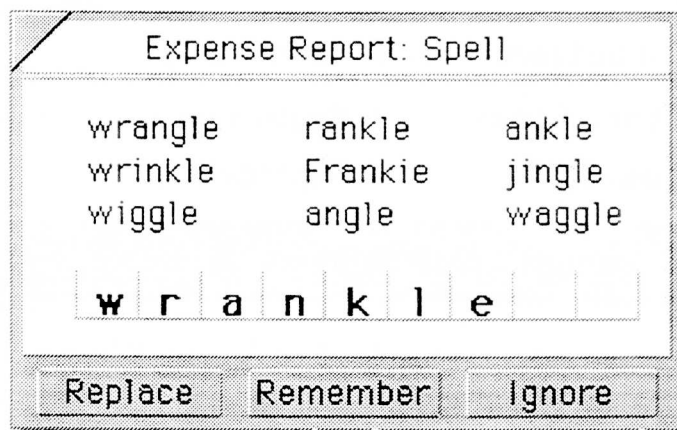
The user taps the **Find** button to initiate the search. When the string in the **Find** field is found, PenPoint selects it (and scrolls it into view if necessary).

The **Replace** button replaces the current selection with the contents of the **Replace** field. The **Replace All** button starts searching from the same place that the **Find** button would, and performs the replace operation each time the target string is found.

⚡ Spell

Choosing **Spell** from the **Edit** menu or making the **S** gesture over the title line begins spell checking from the beginning of the document. Making an **S** gesture over a word in the document selects the word and begins spell checking from that point.

When the spell checker finds a word that is not in either the system dictionary or the current personal dictionary, it displays the **Spell** sheet:



Spell Sheet

The user can:

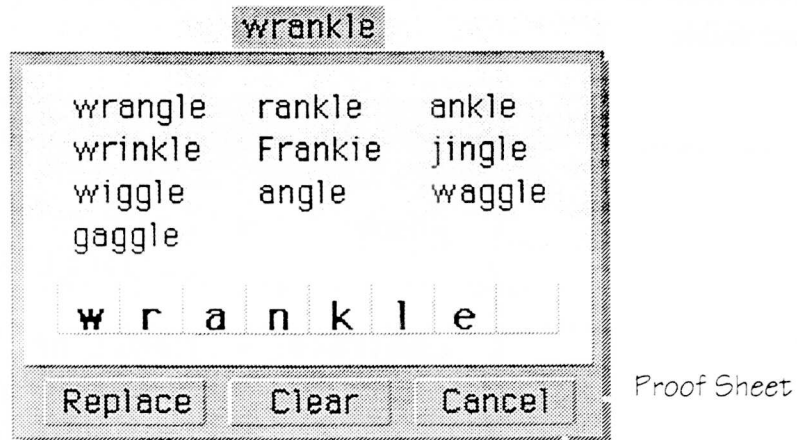
- ◆ Tap one of the alternate words to replace the selected word and continue spell checking.
- ◆ Tap **Replace** to replace the selection with the word in the overwrite field and continue spell checking from just before the replaced word.
- ◆ Tap **Remember** to add the word in the overwrite field to the current user dictionary and continue spell checking.
- ◆ Tap **Ignore** to continue spell checking without changing the selected word.

The **Spell** sheet is modeless, so the user need not dismiss it to interact with the document.

Proof

The **Proof** command, while not on the Edit menu by default, is applicable to any document containing text.

Selecting a word and choosing **Proof** from the **Edit** menu, or making the P gesture over a word, displays the modal **Proof** sheet:



The sheet contains a list of alternative words taken from the dictionary, as well as an overwrite field containing the candidate word. The user can:

- ◆ Tap one of the alternative words to replace the candidate word and dismiss the sheet.
- ◆ Edit the word in the overwrite field, then tap **Replace** to replace the candidate word and dismiss the sheet.
- ◆ Double-tap one of the alternative words to put the tapped-on word in the field, and display a new list of alternatives for the tapped-on word.

Options Menu

The three default document option sheets, accessible from the **Options** menu, are titled **Controls**, **Access**, and **Comments**.

Controls Sheet

The **Controls** sheet allows the user to specify which of the standard document controls are visible:



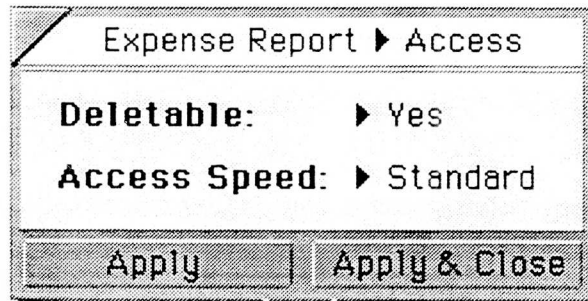
Controls Sheet

Setting **Show** to **Controls** shows the specified controls (and ignores the **Borders** setting). Setting **Show** to **Borders Only** shows only the specified border (and hides all of the controls).

In addition, the user can make the flick up-down gesture in the title line or work area of any document as an accelerator for toggling the state of the **Show** switch. This facilitates working with embedded documents by giving the user a quick way to toggle between the "fully active" form of the document and the "visually clean" form. (See "Compound Documents" in Chapter 2 for an example.)

Access Sheet

The Access sheet is for controls related to accessing the document:



Expense Report ▶ Access

Deletable: ▶ Yes

Access Speed: ▶ Standard

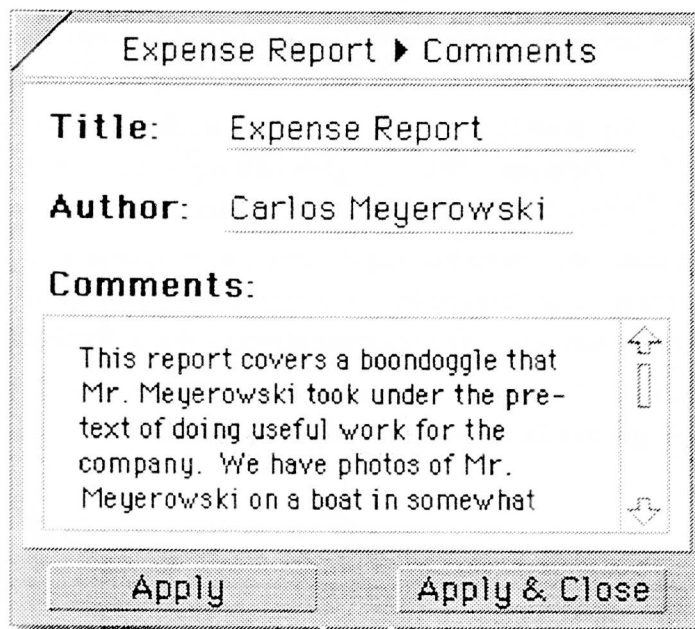
Apply Apply & Close

Access Sheet

The Access Speed control allows the user to make the tradeoff between memory usage and performance. Normally, when the user turns away from a document, the PenPoint operating system files the document's data, freeing up memory for other documents. Setting Access Speed to Accelerated causes PenPoint to keep the document active in memory, so that turning back to it will be quicker (at the expense of using more working storage).

Comments Sheet

The Comments sheet has fields for the title and author, as well as an in-line text box for comments:



Expense Report ▶ Comments

Title: Expense Report

Author: Carlos Meyerowski

Comments:

This report covers a boondoggle that Mr. Meyerowski took under the pretext of doing useful work for the company. We have photos of Mr. Meyerowski on a boat in somewhat

Apply Apply & Close

Comments Sheet

Chapter 7 / Drag & Drop Interface

▼ User Model

The PenPoint™ operating system presents a single user model, called **drag & drop**, for moving and copying objects throughout the environment.

The same drag & drop method works for all types of objects: icons on the Bookshelf, entries in the table of contents, text in a word processor, figures in a graphics editor, cells in a spreadsheet, and so on.

Drag & drop works well regardless of whether the source and destination are in view at the same time. It also handles transferring data across document boundaries and between different types of documents.

▼ Drag & Drop Operation

This section gives a step-by-step description of the drag & drop process.

Step 1. Before dragging the object, the user must indicate whether the desired operation is a move or a copy. The PenPoint operating system provides two ways to do this:

- ♦ **Menu.** The user first selects the object, then chooses **Move** or **Copy** from the Edit menu.
- ♦ **Gesture.** The gesture for move is press: The user touches the object with the pen and pauses for a moment. The copy gesture is tap-press: The user taps the object with the pen, then touches again and pauses.

Step 2. After the user initiates the operation, the PenPoint operating system signals that the object is ready to be dragged by surrounding it with an animated dashed outline, called a **marquee**. Once the marquee appears, the user can drag the object to its destination.

The move marquee is a single outline; the copy marquee is double:



Move & Copy Marquees

The single/double marquee distinction makes it clear to the user throughout the process whether the drag will result in a move or a copy.

Step 3. If the destination is not visible on the screen, the user can lift the pen before dragging. The marquee continues, and the user can then bring the destination into view by scrolling or page turning. Once the destination is on the

screen, the user can complete the drag & drop. If the user begins to drag and then realizes that the destination isn't visible, he or she can simply drop the marquee on the title line, where it will remain. Then the user can bring the destination into view and drag the marquee to it.

The user can cancel the operation after the marquee appears by simply lifting the pen and tapping the selected object to dismiss the marquee. If the user has already begun to drag and wishes to cancel, he or she can do so by dropping the marquee on the title line, then tapping to dismiss it.

Example: Moving a Word in Text

The example below shows the steps in moving a word in MiniText:

Moving a Word in Text

Four score **seven** and years ago. The user first selects the word to be moved.

Four score **seven** and years ago. Then the user initiates the move with the press gesture.

Four score **seven** and years ago. When the user drags the pen, the marquee outline jumps so that the pen tip is centered on the left edge of the outline.

Four score **seven** and years ago. The user points the pen tip directly at the desired location—in the gap between two words.

Four score and **seven** years ago. When the user lifts the pen, the move is completed and the text reformatted. The newly moved word remains selected.

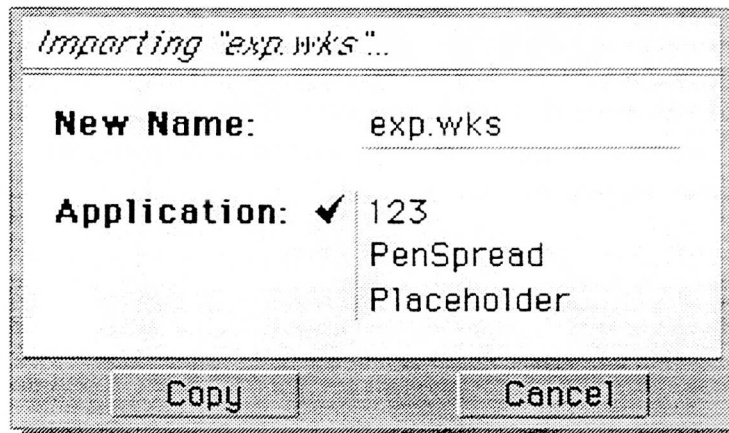
Import and Export

In the PenPoint operating system, **import** means to transfer a non-PenPoint file into the PenPoint environment, associating it in the process with a PenPoint application.

Export means to transfer a PenPoint document or accessory onto a non-PenPoint disk, converting it in the process into a non-PenPoint format.

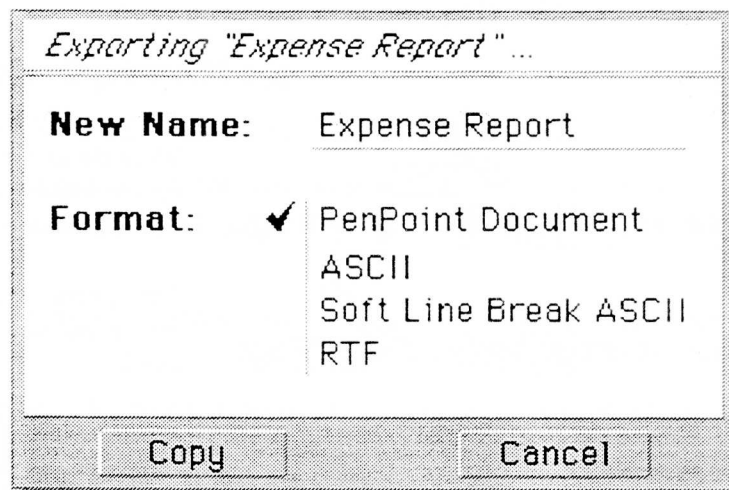
PenPoint doesn't have explicit commands for import or export; instead the user performs these operations using the standard drag & drop interface described in the previous section.

When the user copies a file from a disk into a notebook (or the Bookshelf), a note pops up asking the user to specify an application to associate with the document:



Import Note

Similarly, when the user copies a document from a table of contents to a disk, a note pops up asking for the desired format:

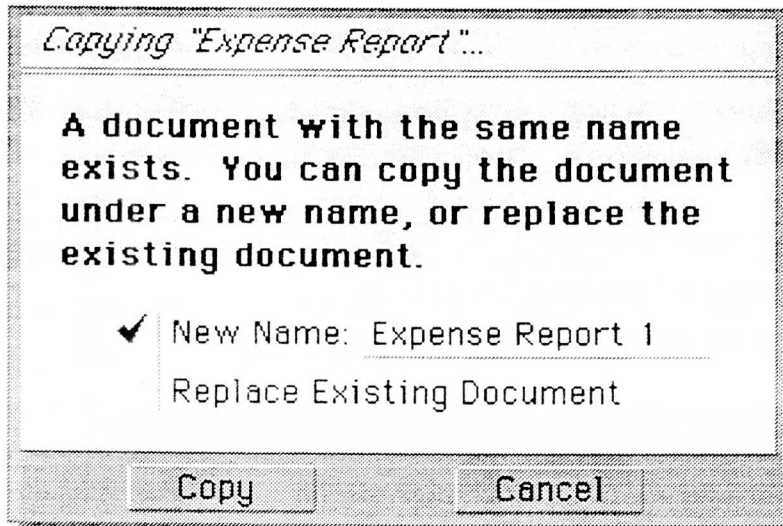


Export Note for Copy

When the import or export operation is a move, the note has both **Move** and **Copy** buttons in order to decrease the likelihood that a user will do a press-drag and mistakenly assume that the document had been copied.

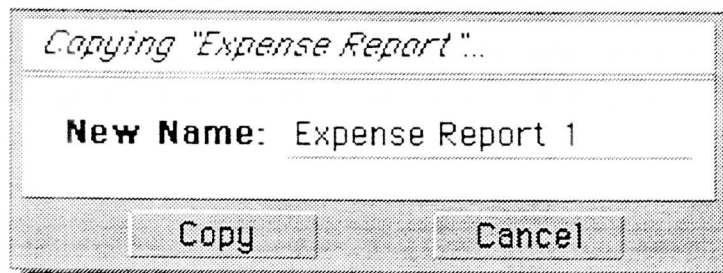
■ Name-Clash Notes

When the user drops a document into a section that already has a document by that name, the note shown below pops up, giving the user the option of creating a new document under a new name or replacing the existing document:



Basic Name-Clash Note

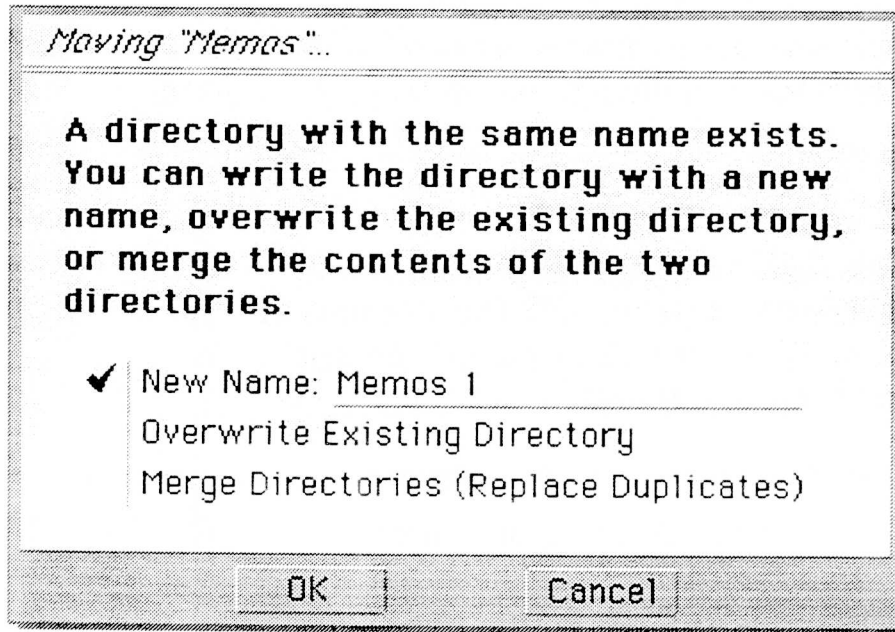
When the user is copying a document within the same section, the note doesn't offer the **Replace Existing Document** option, because that would result in replacing the existing document with itself:



Name-Clash Note for Document Copied Within the Same Section

PenPoint displays name-clash notes for files and directories in the Connected Disks page of the Connections notebook as well as documents and sections in the notebook table of contents.

When a directory name-clash occurs, the note also offers the user the option of merging the contents of the two directories:



Name-Clash Note for
Moving a Directory

Merge Directories means that when both of the identically named directories contain identically named files, the file in the existing directory will be replaced by the file in the directory being moved.

Chapter 8 / Icons

In the PenPoint™ operating system, an icon is a small picture (with an optional label) that suggests the function of an object. Each application has its own icon, supplied by the application developer.

Icon Usage

PenPoint uses icons in many contexts:

- ◆ Icons are used to represent open and closed sections as well as individual documents in the notebook table of contents.
- ◆ Each accessory is represented as an icon in the Accessories palette.
- ◆ The Bookshelf holds icons representing notebooks, documents, and accessories.
- ◆ Icons are used to represent disks at the top of the Connected Disks page in the Connections notebook. In the display of disk contents, icons are used to represent open and closed directories, non-PenPoint files and PenPoint documents, and the various types of installable resources such as fonts, handwriting engines, and user dictionaries.
- ◆ The user can embed documents within other documents, either in the work area or in the cork margin. An icon is shown whenever an embedded document is closed.

▼ Response to Gestures

Icons respond to several gestures, including press to move, tap-press to copy, cross-out to delete, check for options, circle to edit the label, and plus to select.

Response to the tap and double-tap gestures depends on the icon's context:

- ◆ **Table of Contents.** Tapping on the picture part of the icon turns to the notebook page containing the document; tapping on the label selects the document. Double-tapping on the picture floats the page in front of the table of contents; double-tapping on the label of a section expands or collapses the display of the section's contents.
- ◆ **Disk Directory.** Tapping anywhere on the icon, picture, or label selects the object.
- ◆ **Bookshelf and Cork Margin.** Tapping or double-tapping anywhere on the icon opens the document.
- ◆ **Document Work Area.** Tapping anywhere on the icon opens the document. Whether the opened document is displayed floating on top of the container document or as part of the container document's scrollable surface depends on the settings of the **Open** control on the icon's option sheet, described later in this chapter.

▼ Open and Closed Appearance

When an icon is open, its outline is filled in with light grey. This convention allows the user to tell at a glance when an icon is open.

In the figure below, Settings, Accessories, and the Keyboard are open; the other icons are closed:



▼ Icon Sizes

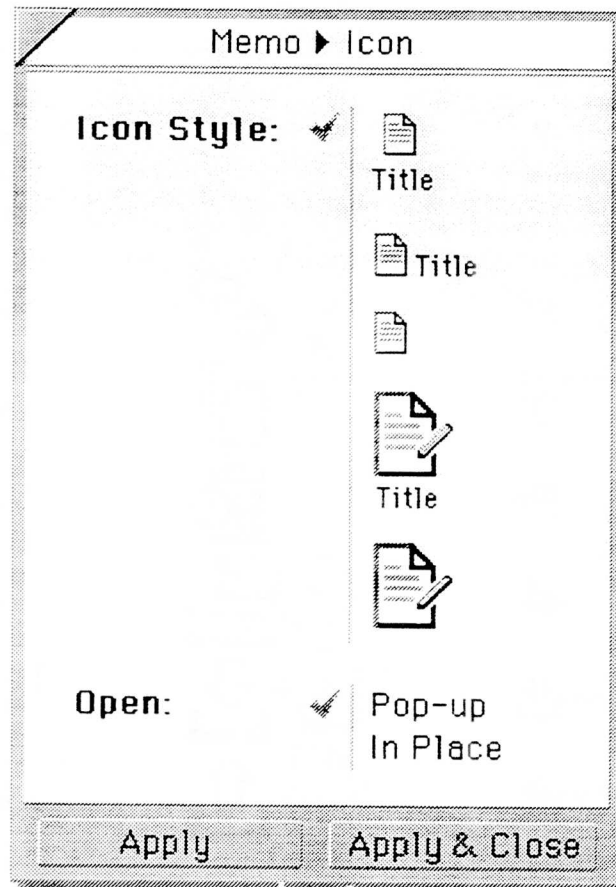
Application developers provide two bitmaps for each icon: one for each of the two sizes of icons. The sizes are:

- ◆ **Small.** 16 pixels high; no more than 26 pixels wide.
- ◆ **Large.** 30 pixels high; no more than 44 pixels wide.

The application developer also specifies a **hot point** for the icon. PenPoint uses the hot point to line up the icons correctly in the table of contents.

Icon Option Sheet

In most contexts, making the check gesture over an icon displays the standard icon option sheet:



Icon Option Sheet

Icons are treated differently in different contexts:

- ◆ The notebook table of contents and the directory view of the Connected Disks page in the Connections notebook do not present the icon option sheet to the user. In those contexts, the icon style is always a small picture followed by a label.
- ◆ The Bookshelf, cork margin, and Connected Disks page of the Connections notebook restrict all their icons to the same style. Changing the style on one icon in these regions causes all the icons to change.
- ◆ The **Open** control is dimmed when the icon is in the Bookshelf or cork margin, which restrict icons to opening **Pop-up**.

Standard PenPoint Icons

The following table lists all of the icons used in the PenPoint operating system.

PenPoint Icons

Bookshelf Icons

		Help
		Settings
		Connections
		Stationery
		Accessories
		Keyboard
		In box
		Out box

Notebook Icons

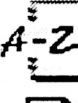
		Notebook
		Section (Closed)
		Section (Open)

PenPoint Icons (Continued)

Document Icons

		Default Document
		MiniText
		MiniNote
		GO Fax
		Placeholder

Accessory Icons

		Clock
		Address Book
		Bitmap Editor
		System Log

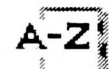
Disk and Printer Icons

		Hard Disk
		Floppy Disk
		Network Disk
		Printer

Disk Contents Icons

		Folder (Closed)
		Folder (Open)
		File

Installable Software Icons

		Application
		Service
		Writing
		Gesture
		Font
		Dictionary
		User Profile

Chapter 9 / Reference Buttons

The user can create reference buttons for quick navigation to any selectable object. The buttons can be placed in any region that accepts embedded objects.

▼ Creating Reference Buttons

The user creates a reference button by first selecting the destination, then making the double-circle gesture to create the button.

If the selection is a document in the table of contents, the button will have the document's title as its label. If the selection is a text span, the label will be the first word of the selection.

▼ Response to Gestures

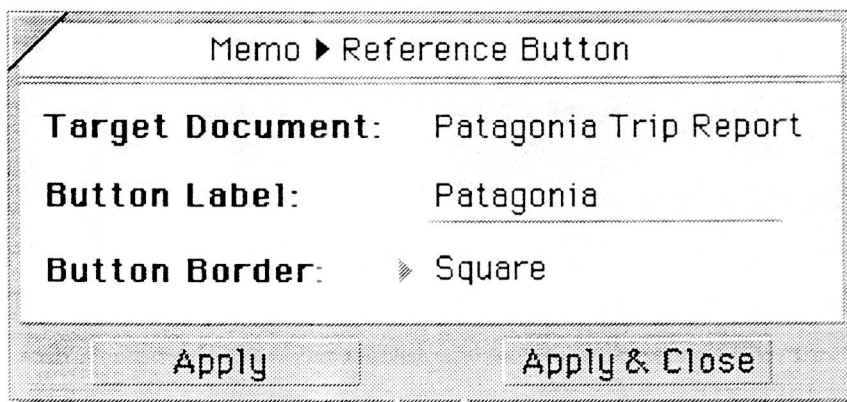
Reference buttons respond to the same set of gestures as icons: tap and double-tap to navigate, press to move, tap-press to copy, cross-out to delete, check for options, circle to edit the label, and plus to select.

Tapping on a button brings the referenced document into view. If the referenced document is in a notebook, the PenPoint™ operating system opens the notebook if necessary, then turns to the document and scrolls to the location referenced by the button. If the destination is an accessory in the Accessories sheet, PenPoint opens the accessory directly, without opening the Accessories sheet.

Double-tapping on a button floats the document and scrolls to the referenced location.

Reference Button Option Sheet

Like all objects in the PenPoint operating system, reference buttons have their own option sheet:

A screenshot of a dialog box titled "Memo ▶ Reference Button". The dialog box has a title bar with a diagonal line in the top-left corner. It contains three labeled text fields: "Target Document:" with the value "Patagonia Trip Report", "Button Label:" with the value "Patagonia", and "Button Border:" with a dropdown menu showing "▶ Square". At the bottom of the dialog box are two buttons: "Apply" and "Apply & Close".

Memo ▶ Reference Button	
Target Document:	Patagonia Trip Report
Button Label:	Patagonia
Button Border:	▶ Square
Apply Apply & Close	

Reference Button
Option Sheet

Chapter 10 / Busy Clock

Whenever the input system is not responsive to user input because the system is busy, the PenPoint™ operating system displays a small **busy clock**:



The hands of the busy clock tick through eight positions, one tick every quarter of a second.

The input system automatically displays the busy clock after a short pause at the location of the last pen-up event. Applications can override this default behavior by displaying the busy clock right away, suppressing the display, or changing the location.

Part 4

System Applications

THE PENPOINT USER INTERFACE

PART 4 / SYSTEM APPLICATIONS

Chapter 11 / Notebook Table of Contents

Menus	85
Document Menu	85
Edit Menu	86
View Menu	86
Create Menu	86
Option Sheets	87
Layout Sheet	88
Document Sheet	89

Chapter 12 / Settings Notebook

Overview	90
Contents	90
Preferences	91
Writing	92
Pen	92
Fonts & Layout	93
Float & Zoom	94
Date	94
Time	95
Sound	95
Power	96
Installed Software	97
Overview	97
Functions Common to All Categories	98
Handwriting	99
User Dictionaries	100
Fonts	101
User Profiles	101
Status	102
Storage Summary	102
Storage Details	103
PenPoint	104

Chapter 13 / Connections Notebook

Overview	105
Contents	105
Connected Disks	106
Menus	107
Views	108
Option Sheets	111
Quick Installer	113
Network Disks	114
Menus	114
Option Sheets	115
Connected Printers	116

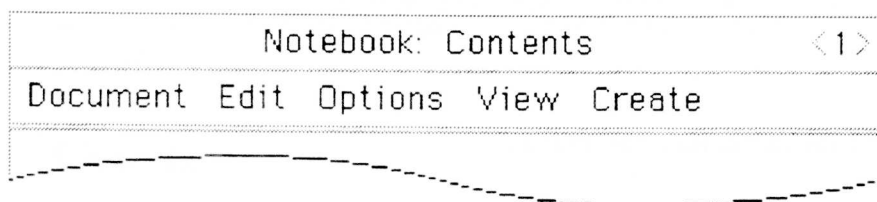
Menus	116
Option Sheets	117
Network Printers	118
Menus	119
Option Sheets	119
Chapter 14 / Stationery Notebook	120
Chapter 15 / In Box and Out Box	121
In Box	121
Out Box	122
Chapter 16 / Help	123
Quick Help	123
Help Notebook	124

Chapter 11 / Notebook Table of Contents

The main table of contents appears on the first page of the notebook. Each section in the notebook has its own table of contents on a separate page, immediately preceding the pages in the section.

Menus

The table of contents menu line has five menus:



Menu Line for Table of Contents

Document Menu

The Document menu for the table of contents has only two commands:

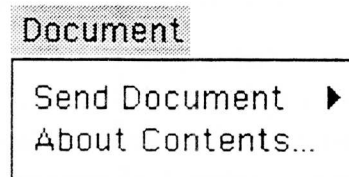


Table of Contents
Document Menu

Send Document brings up a submenu showing all the installed Out box services. About Contents brings up information sheets pertaining to the table of contents itself.

✦ Edit Menu

The table of contents Edit menu has four commands:

Edit

Move...
Copy...
Delete...
Rename...

Table of Contents Edit
Menu

NOTE: There is no **Select All** command because the table of contents does not support the selection of multiple documents for Release 1.0. Multiple selection, as well as standard editing commands such as **Undo**, **Find**, and **Spell**, will be implemented in future releases of the PenPoint™ operating system.

✦ View Menu

The table of contents View menu has four commands:

View

Expand
Collapse
.....
Turn To
Bring To

Table of Contents View
Menu

Expand displays the names of the documents and sections in the next level of the selected section (if no section is selected, all sections are expanded one level).

Collapse hides all documents and sections in the selected section (if no section is selected, then all sections are collapsed).

Turn To turns to the selected document; **Bring To** floats the selected document.

✦ Create Menu

The **Create** menu allows the user to create new documents and new sections.

The user specifies which documents appear on the **Create** menu by checking the desired documents in the Stationery notebook, as described in Chapter 14.

The figure below shows a typical Create menu:

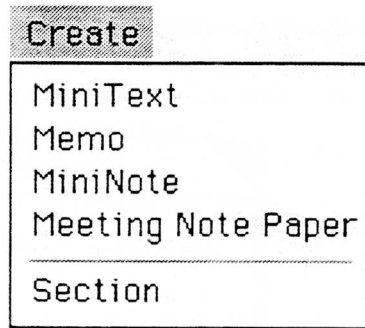


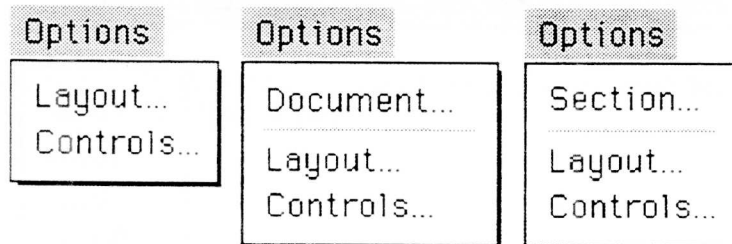
Table of Contents
Create Menu

The newly created document or section appears in the notebook immediately after the selected document or section, or at the end of the notebook if there is no selection.

Instead of first making a selection and then tapping on **Create** in the menu line, the user can bring up the menu at the desired location in the table of contents with the caret gesture.

Option Sheets

As described in Chapter 7, the contents of the **Options** menu changes with the selection. The options menu for the main table of contents has three states:



Options Menu for Main
Table of Contents

The menu on the left implies that there is no selection within the table of contents. The center menu implies that a document is selected. The menu on the right implies that a section is selected.

NOTE: The main table of contents doesn't have an **Access** option sheet because it is not deletable and its **Access Speed** is always set to **Accelerated**. Nor does it have a **Comments** option sheet. All other table of contents pages do have **Access** and **Comments** option sheets.

⚡ Layout Sheet

The **Layout** option sheet allows the user to modify the layout of the table of contents:

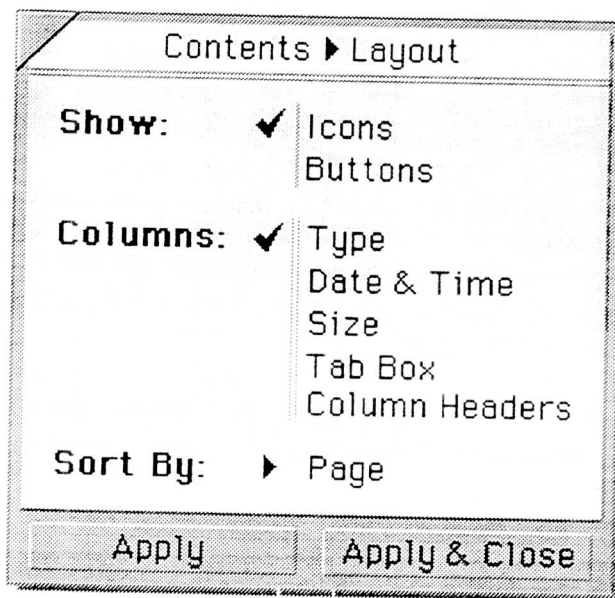


Table of Contents
Layout Sheet

The **Show** list is unusual in that tapping on the check causes it to disappear. This allows the user to choose **Icons**, **Buttons**, or neither.

The **Sort By** menu allows the user to sort on the various document attributes:

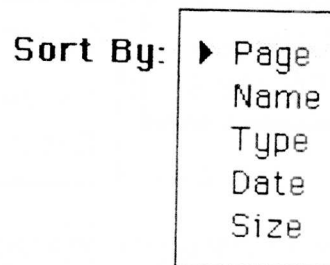
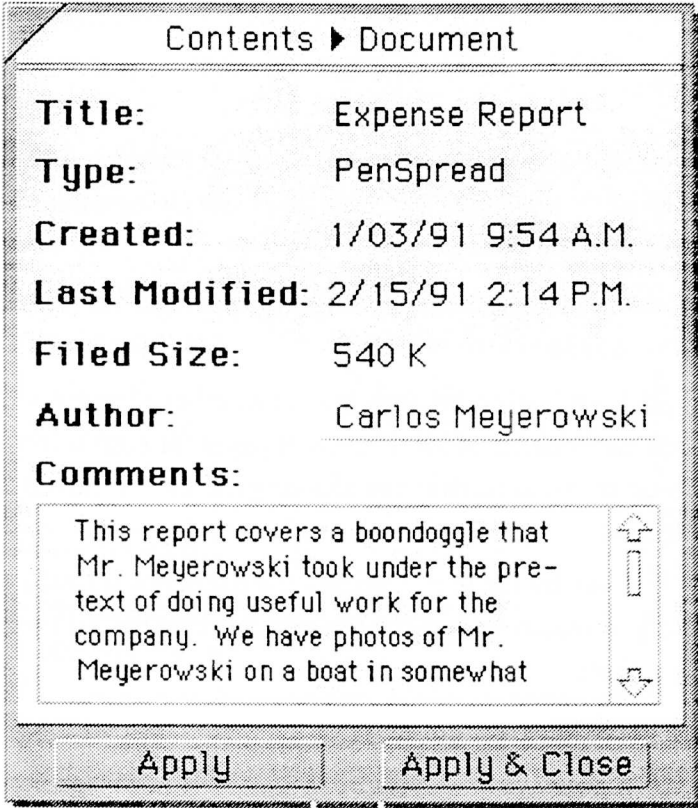


Table of Contents Sort
Menu

Document Sheet

Making a check on the title of a document brings up the following option sheet:



The image shows a dialog box titled "Contents ► Document". It contains the following information:

Title:	Expense Report
Type:	PenSpread
Created:	1/03/91 9:54 A.M.
Last Modified:	2/15/91 2:14 P.M.
Filed Size:	540 K
Author:	Carlos Meyerowski
Comments:	This report covers a boondoggle that Mr. Meyerowski took under the pretext of doing useful work for the company. We have photos of Mr. Meyerowski on a boat in somewhat

At the bottom of the dialog box are two buttons: "Apply" and "Apply & Close".

Option Sheet for
Document in Table of
Contents

The option sheet for a section is identical except for the title, which reads Section instead of Document.

Chapter 12 / Settings Notebook

Overview

The Settings notebook gives the user a single place to go to view and modify settings both for the PenPoint™ operating system as a whole and for whatever software is currently installed in the system.

While it employs the notebook metaphor, the Settings notebook differs from the normal PenPoint data notebook in several ways:

- ♦ It is optimized for quick navigation through a small number of pages with only one level of sections. Instead of local contents pages for each section, each page has a pop-up menu in its title line allowing the user to turn to any other page in that section.
- ♦ It is not editable in any way by the user. Neither the pages nor the tabs can be deleted, re-ordered, or renamed, and the table of contents has no menu line and no display options.
- ♦ It doesn't have page numbers.

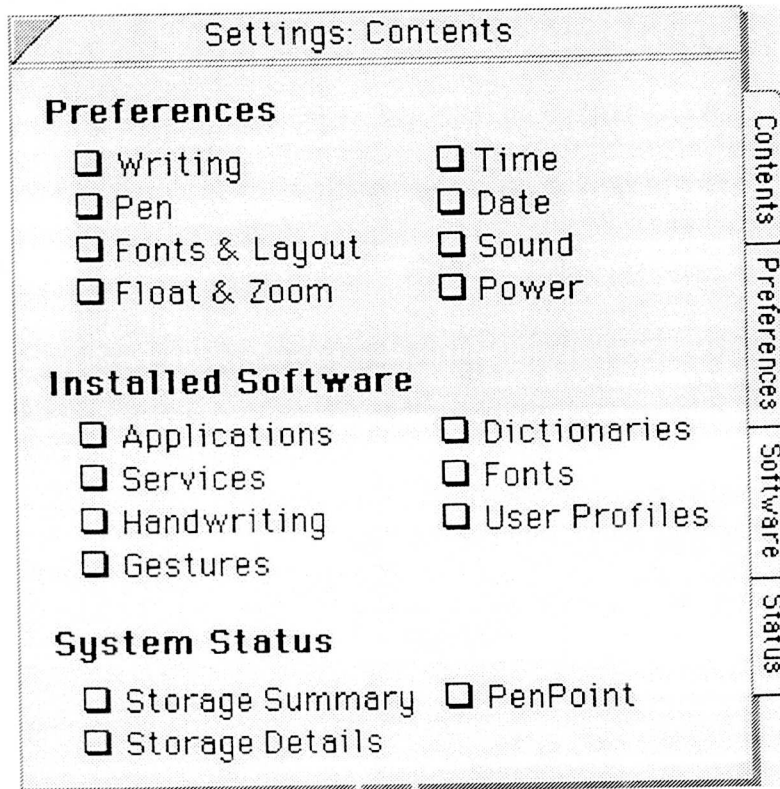
This form of notebook is also used for the Connections notebook, described in Chapter 13.

Contents

The Settings notebook is organized into three sections:

- ♦ **Preferences.** For setting system-wide user preferences.
- ♦ **Installed Software.** There is a page for each category of installed software. For categories representing system-wide resources used by PenPoint (Handwriting, Gestures, Dictionaries, Fonts, and User Profiles), the page includes a column of checkboxes for the user to specify which installed resource is currently being used.
- ♦ **Status.** For system status.

Unlike other notebooks, the Settings notebook always opens to its table of contents:

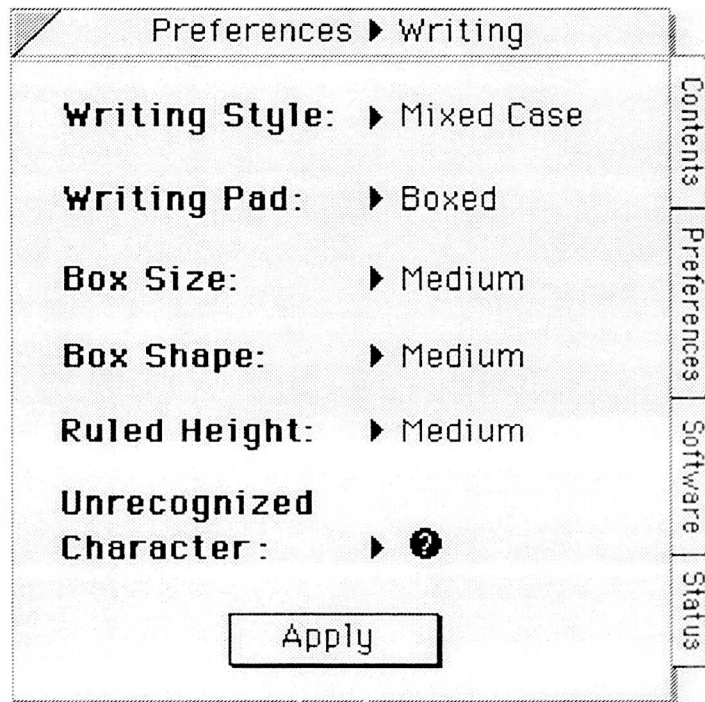


Settings Notebook
Contents

▼ Preferences

There is a separate page in the Settings notebook for each category of user preference.

The next several pages contain figures showing each of the preference pages.

 Writing

Preferences ▶ Writing

Writing Style: ▶ Mixed Case

Writing Pad: ▶ Boxed

Box Size: ▶ Medium

Box Shape: ▶ Medium

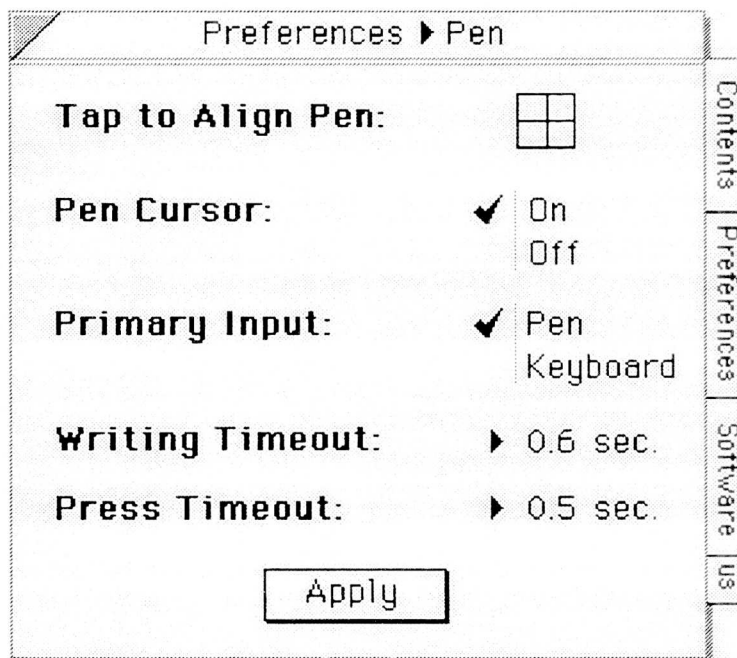
Ruled Height: ▶ Medium

Unrecognized Character: ▶ ?

Apply

Contents Preferences Software Status

Writing Preferences

 Pen

Preferences ▶ Pen

Tap to Align Pen: 

Pen Cursor: ☒ On
☐ Off

Primary Input: ☒ Pen
☐ Keyboard

Writing Timeout: ▶ 0.6 sec.

Press Timeout: ▶ 0.5 sec.

Apply

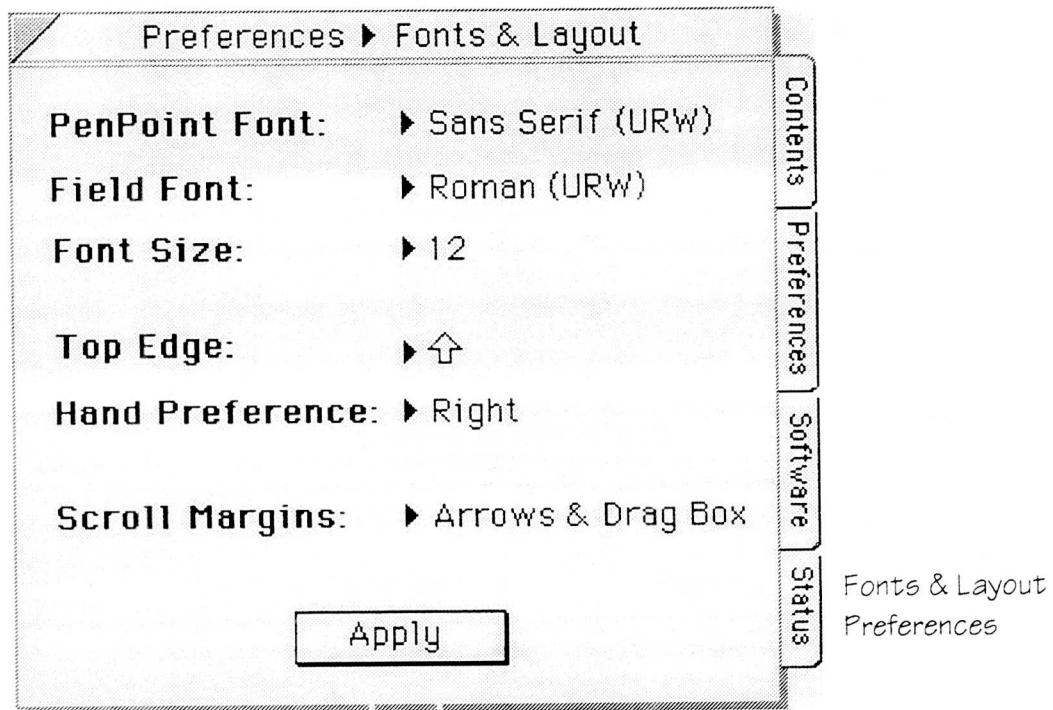
Contents Preferences Software Status

Pen Preferences

The **Writing Timeout** is the interval the PenPoint operating system pauses after the user lifts the pen from the screen before translating the pen strokes into gestures or handwriting.

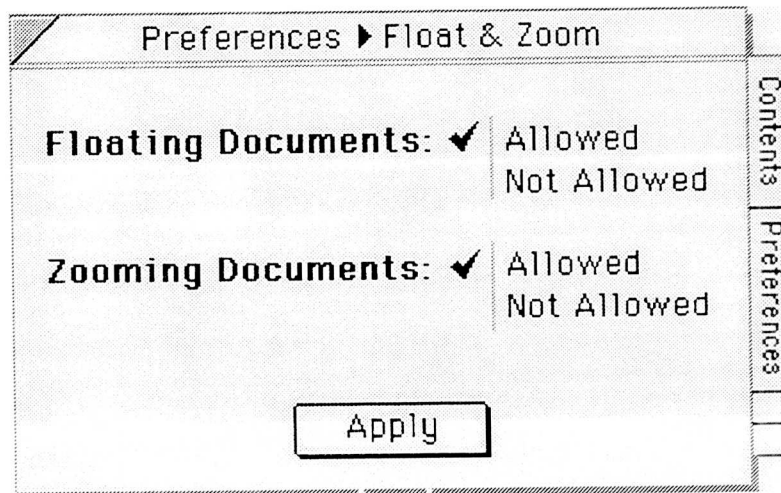
The **Press Timeout** determines how long the user must touch the pen to the screen before the press gesture is recognized.

⚡ Fonts & Layout



The menu for **Top Edge** has four arrows, one pointing to each edge of the computer. The user chooses the appropriate arrow to re-orient the display such that that edge is at the top. After the user taps the Apply button, the arrow again points to the top edge.

The **Scroll Margins** control allows the user to choose either a traditional scroll margin with arrows and a drag box, or a simple margin for flicking. See the section on “Scroll Margins” in Chapter 4 for more details.

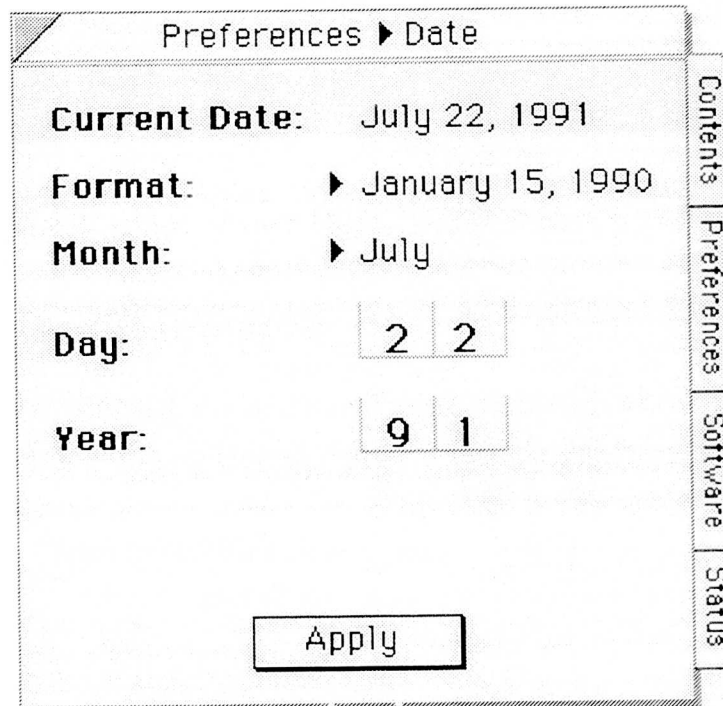
 Float & Zoom

Preferences > Float & Zoom

Floating Documents: ☒ Allowed ☐ Not Allowed

Zooming Documents: ☒ Allowed ☐ Not Allowed

Apply

Float & Zoom
Preferences **Date**

Preferences > Date

Current Date: July 22, 1991

Format: ▶ January 15, 1990

Month: ▶ July

Day: 2 2

Year: 9 1

Apply

Date Preferences

 Time

Preferences ▶ Time

Current Time:	9:20 A.M.
Time Zone:	▶ Los Angeles (-8)
Format:	▶ 12 Hour
Seconds:	▶ Not Displayed
Hour:	
Minute:	
Second:	
A.M./P.M.	▶ A.M.

Apply

Contents
Preferences
Software
Status

Time Preferences

 Sound

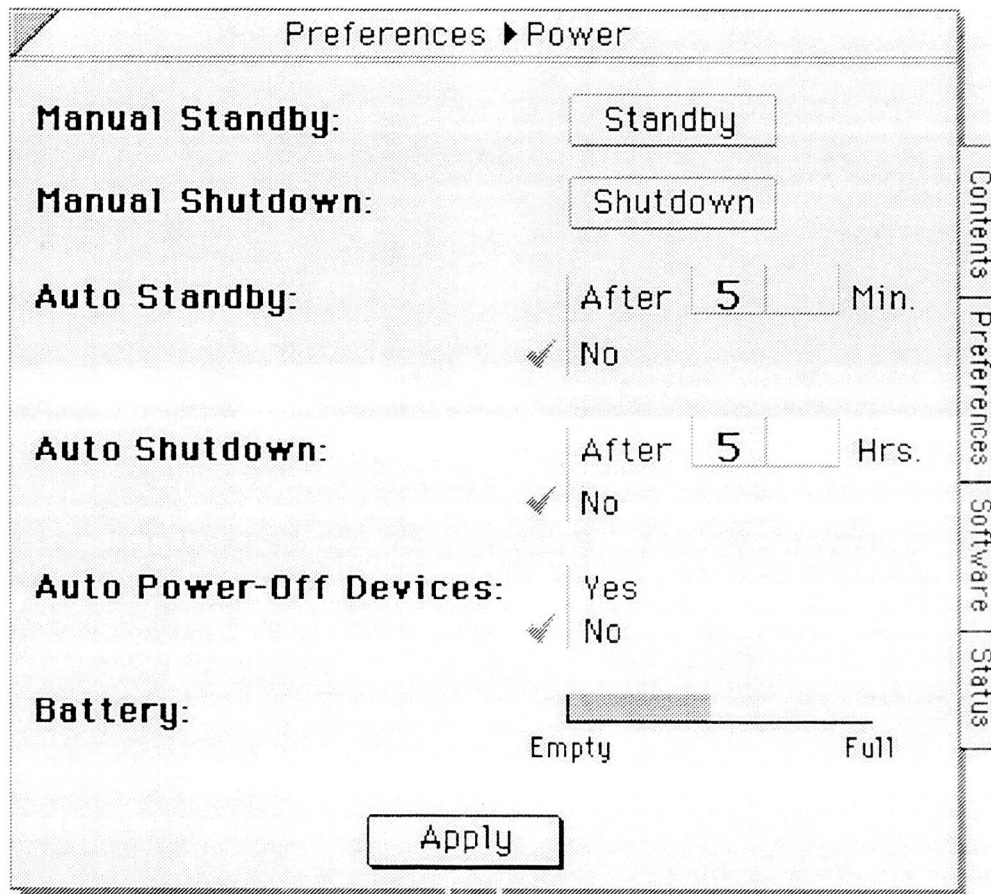
Preferences ▶ Sound

Warning Beep:	☒ On ☐ Off
---------------	--

Apply

Contents
Preferences

Sound Preferences

 Power

Preferences > Power

Manual Standby: Standby

Manual Shutdown: Shutdown

Auto Standby: After 5 Min.
☒ No

Auto Shutdown: After 5 Hrs.
☒ No

Auto Power-Off Devices: Yes
☒ No

Battery: 
Empty Full

Apply

Power Preferences

Tapping the **Standby** button puts the processor into suspend mode.

Tapping the **Manual Shutdown** button shuts the system down—if **Auto Standby** is on, the processor goes into suspend mode. Otherwise, it is turned off completely.

If **Auto Standby** is on, PenPoint will automatically suspend the processor whenever there is no input from the user for the specified period. When the processor is suspended, the screen goes blank. Pressing the power button on the computer awakens the processor and restores the display.

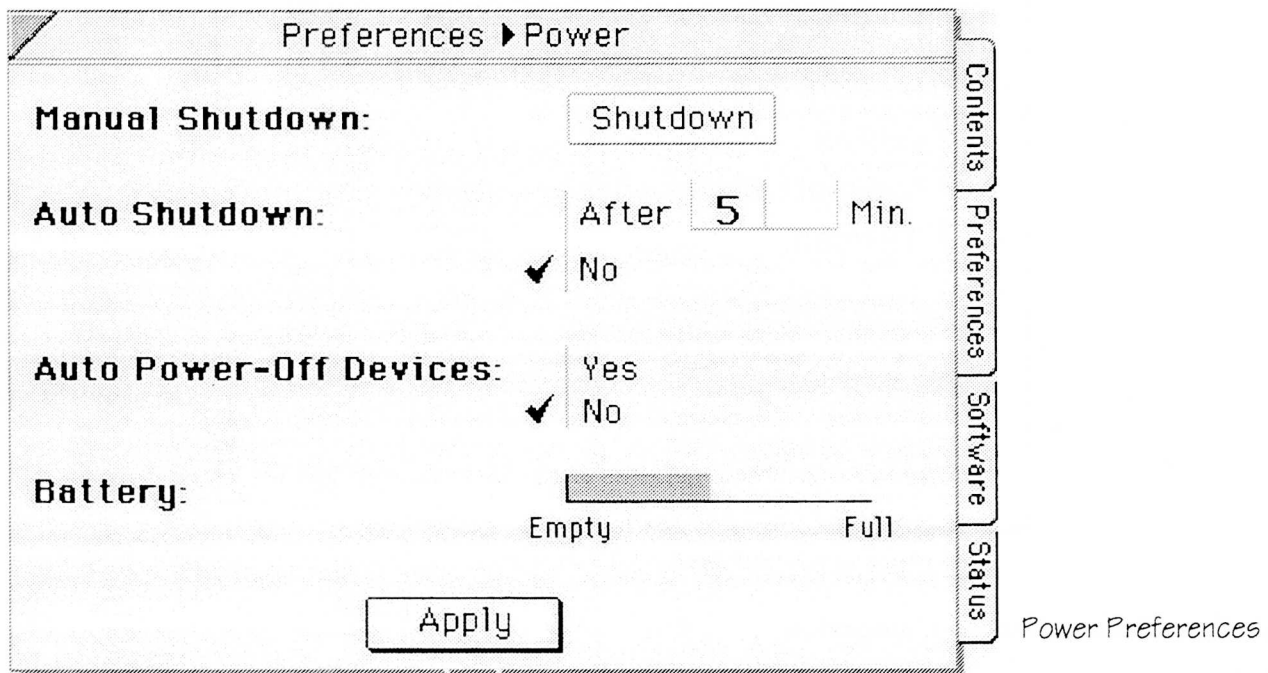
If **Auto Shutdown** is on, PenPoint will automatically shut the processor down whenever the system remains suspended for the specified period.

Setting **Auto Power Off Devices** to **Yes** causes both the computer's main circuitry and the various peripherals—disks, keyboard, and so on—to automatically power down when not in use. The hardware manufacturer specifies the period of inactivity after which each device powers down.

When any peripheral has powered down, the Settings icon “droops,” so the user can tell at a glance when anything is not active. The next figure shows the two forms of the Settings icon:



Triple-tapping on the dimmed icon powers up any peripherals that have powered down. In the event that the machine does not have the suspend capability, the Power sheet appears as in the following figure:



There is no Auto Suspend control, and the effect of the Auto Shutdown control is different. If Auto Shutdown is on, the PenPoint operating system will automatically turn the machine off whenever there is no input from the user for the specified period.

Tapping the Manual Shutdown button has the same effect.

Installed Software

The second section of the Settings notebook is for Installed Software.

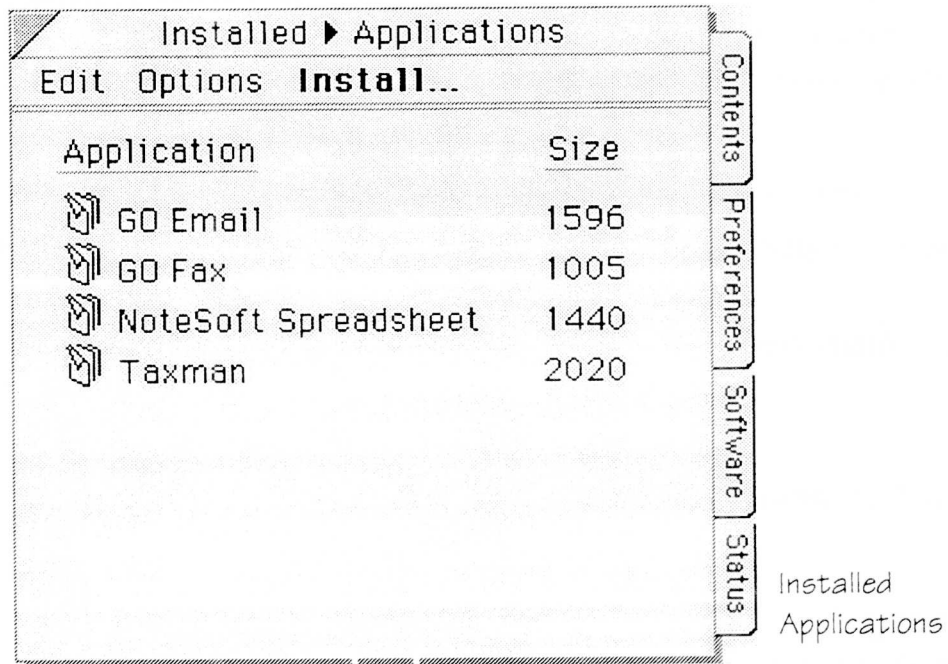
Overview

The Installed Software section has a page for each category of installable software. From these pages the user can:

- ◆ See what's currently installed on the machine.

- ◆ Perform housekeeping functions such as deinstalling, saving to disk, restoring from disk, or deactivating a particular piece of software.
- ◆ Set any options that the installed software provides. For example, an application may provide options that apply to all instances of the application.

The Applications page is typical of the Installed Software pages:

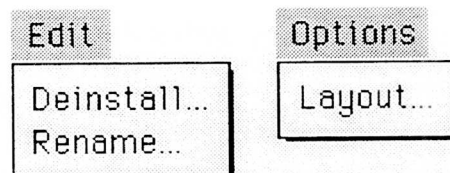


Functions Common to All Categories

While each of the pages in the Installed Software section is tailored to the specific category of installable software, the core functionality is common across categories.

Menus

All categories have the same two menus:



Installed Software
Menus

Install Sheet

The **Install...** button at the right of the menu line displays a sheet showing the installable software of the appropriate category:

Installable Applications		
Hard Disk 1		
Disk	Edit Options	Hard Disk 1
Application	Version	Install
 GO Write	1.0	☐
 GO Draw	2.0	☒
 GO Figure	1.0	☐

Install Sheet

The user simply taps on the checkmark to install (or deinstall) any of the installable items.

Handwriting

The figure below shows the Installed Handwriting page:

Installed ► Handwriting					
Edit	Options	Install...	Customize...	Practice...	
Handwriting	Type	Size	Modified?	Current	
 Standard	GO	1596	No	☐	
 Arnold	GO	1005	No	☐	
 Zoot	GO	1440	No	☒	
 Mitch	GO	2020	No	☐	
 TrueWrite	TrueWrite	1668	No	☐	

Installed
Handwriting

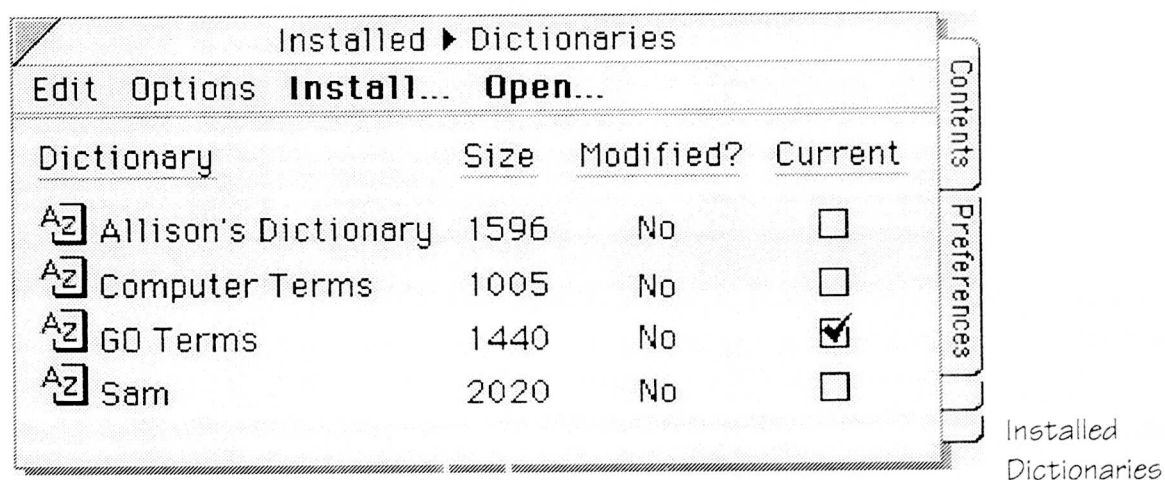
The **Type** column shows what handwriting engine the profile is associated with.

The **Customize...** and **Practice...** buttons bring up sheets that allow the user to train or practice on the selected handwriting profile. If no profile is selected, the buttons apply to the current profile.

NOTE: The training and practice utilities may or may not be present for a given handwriting profile. If either is not present for the current profile, the corresponding button is dimmed.

✦ User Dictionaries

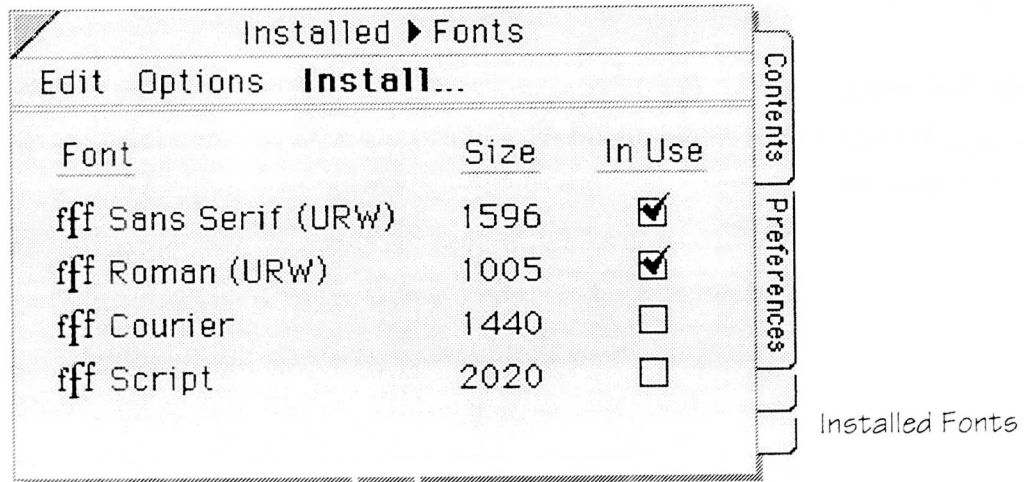
The figure below shows the Installed Dictionaries page:



Tapping **Open...** opens the selected dictionary (or current dictionary, if none is selected) into its own pop-up sheet.

Fonts

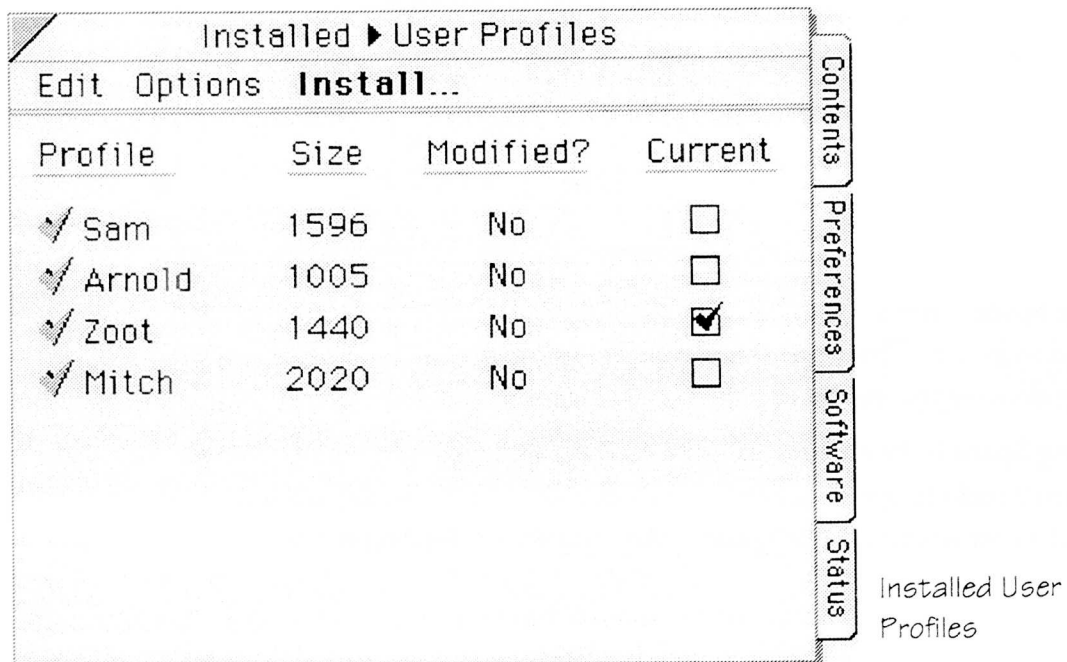
The figure below shows the Installed Fonts page:



Font	Size	In Use
fff Sans Serif (URW)	1596	☒
fff Roman (URW)	1005	☒
fff Courier	1440	☐
fff Script	2020	☐

User Profiles

The figure below shows the Installed User Profiles page:



Profile	Size	Modified?	Current
✓ Sam	1596	No	☐
✓ Arnold	1005	No	☐
✓ Zoot	1440	No	☒
✓ Mitch	2020	No	☐

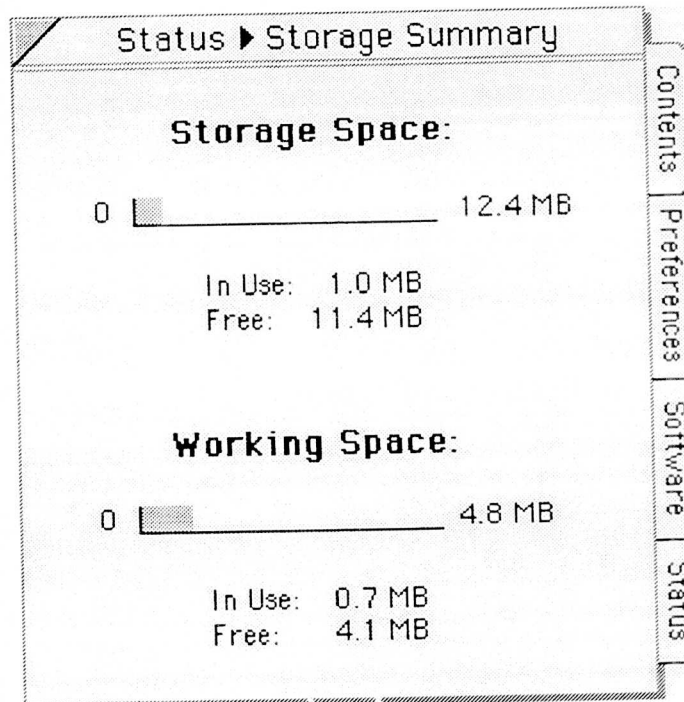
Each user profile includes all the preference settings for that user. Tapping the checkbox to make the profile current has the effect of applying all the user's preference settings in a single step. By allowing the user to customize many aspects of the environment in a single step, the user profile facility makes it easy for more than one person to share a single PenPoint machine.

▼ Status

The Status section of the Settings notebook contains three pages: Storage Summary, Storage Details, and PenPoint.

▼ Storage Summary

The Storage Summary sheet shows the user how much space is currently used, and how much remains to create and work with documents:



Storage Summary
Sheet

Storage Space is the amount of space available for documents, accessories, and installed software. This is the space available to create new documents, copy existing documents, import files from disk, or install new software.

Working Space is the amount of space available for active documents. Active documents include open documents and accessories, and also any documents for which the user has set **Access Speed** to **Accelerated** on the **Access** sheet.

⚡ Storage Details

The user may have questions that cannot be answered in the terms of the simple storage model presented in the Storage Summary sheet. Typical questions include:

- ◆ How much RAM capacity does the machine have?
- ◆ How big is the internal disk?
- ◆ How much storage does PenPoint occupy?
- ◆ How much storage space is devoted to installed software? To documents?

The user can find the answers to these questions by referring to the Storage Details sheet:

Status ▶ Storage Details

Contents

Preferences

Software

Status

Configuration:

Installed RAM:

8.0 MB

Internal Disk:

20.0 MB

Disk Usage:

PenPoint (System):

1.6 MB

PenPoint (Other):

4.0 MB

Working Reserve:

2.0 MB

Installed Software:

0.5 MB

Documents:

0.5 MB

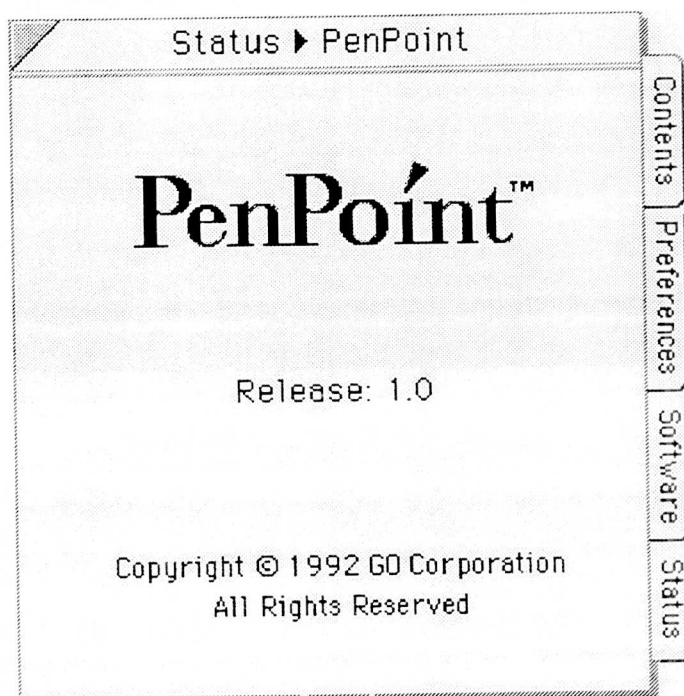
Free:

11.4 MB

Storage Details
Sheet

 PenPoint

The PenPoint status sheet shows which version of PenPoint is installed:



PenPoint Sheet

Chapter 13 / Connections

Overview

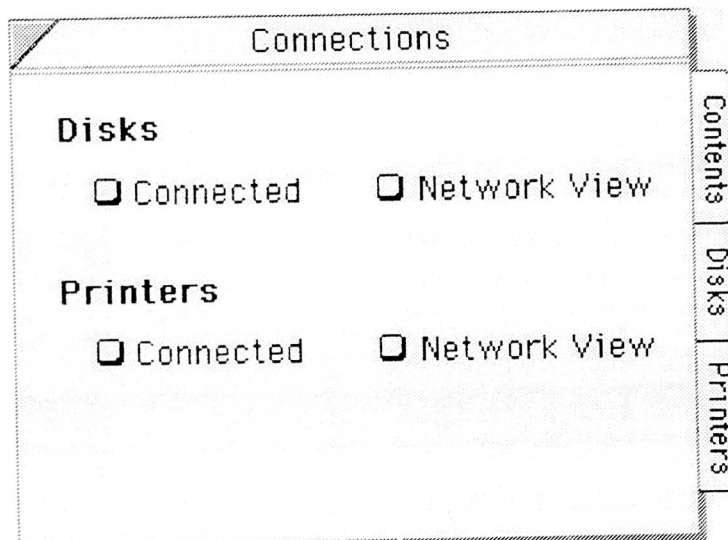
The Connections notebook is a utility notebook, like the Settings notebook described in the preceding chapter.

The Connections notebook gives the user a single point of access for all connections to the world outside the computer. Through Connections, the user can:

- ◆ Import files, transforming them into PenPoint documents.
- ◆ Export documents, transforming them into non-PenPoint files.
- ◆ Install software from disk.
- ◆ Set up printers.
- ◆ Browse disks and perform disk management functions (move, copy, delete and rename files, format disks, and so on).
- ◆ Browse networks and make network resources (disks and printers) available for use.

Contents

The Connections notebook is organized into two sections: one for disks and one for printers:

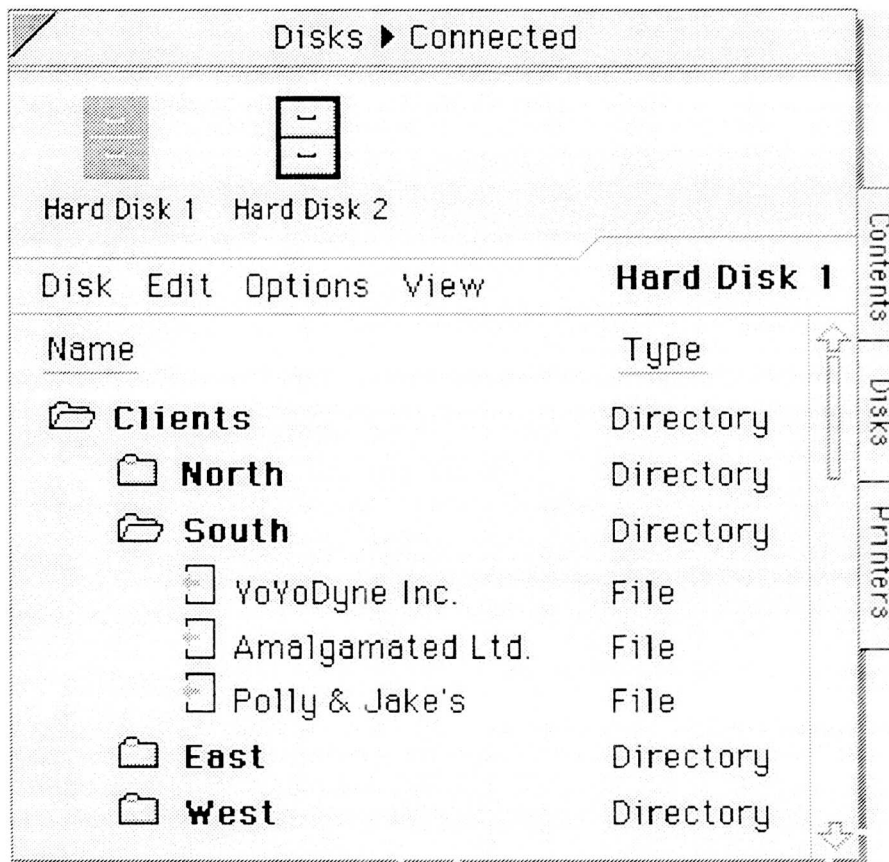


Connections
Notebook Table of
Contents

Connected Disks

The Connected Disks page allows the user to access all disks that are currently connected. Such disks may be built into the machine, connected directly through a port, or connected indirectly over a network.

An icon for each connected disk appears along the top. When the user taps the icon, a card showing the disk's contents opens up. In the example below, there are two connected disks: Hard Disk 1 and Hard Disk 2. The user has opened Hard Disk 1:



Connected Disks
Page of Connections
Notebook

Menus

Instead of a single menu line at the top of the page, there is a menu line for each open disk. Each menu line contains the following menus:

Disk Close Duplicate Update Display Format...	Edit Move... Copy... Delete... Rename... Create Directory...
Options Layout... Disk... Disk Icons...	View ☒ Directory Applications Services Handwriting Gestures Dictionaries Fonts User Profiles Bookshelf Expand Collapse

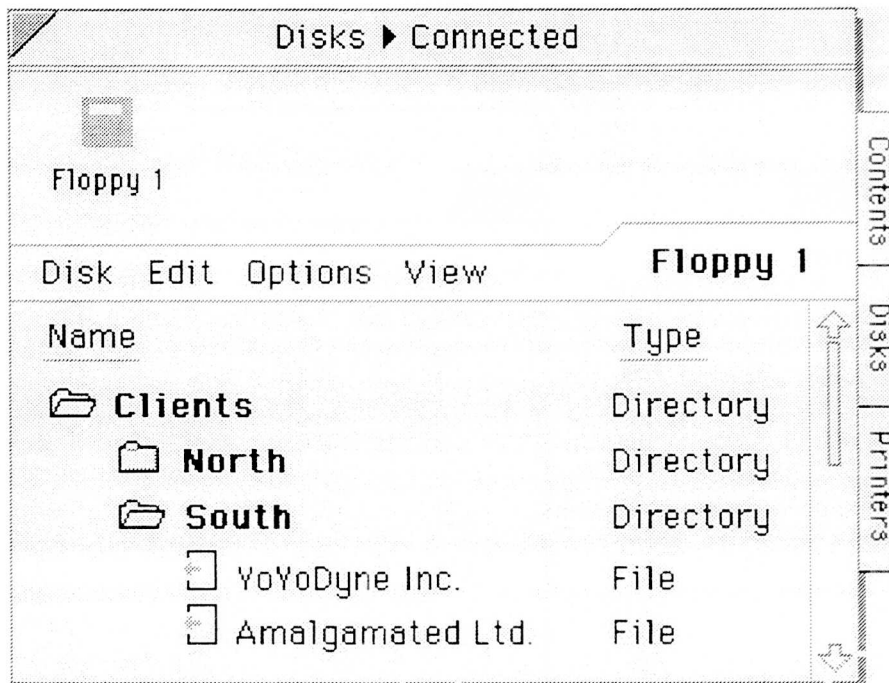
Connected Disk
Menus

Views

The **View** menu allows the user to choose how to view the contents of the disk. The views fall into three categories: Bookshelf, Directory, and Installable Software. Each of these views has its own purpose.

Directory View

The default Directory view is the familiar PC type of nested display with directories and files:



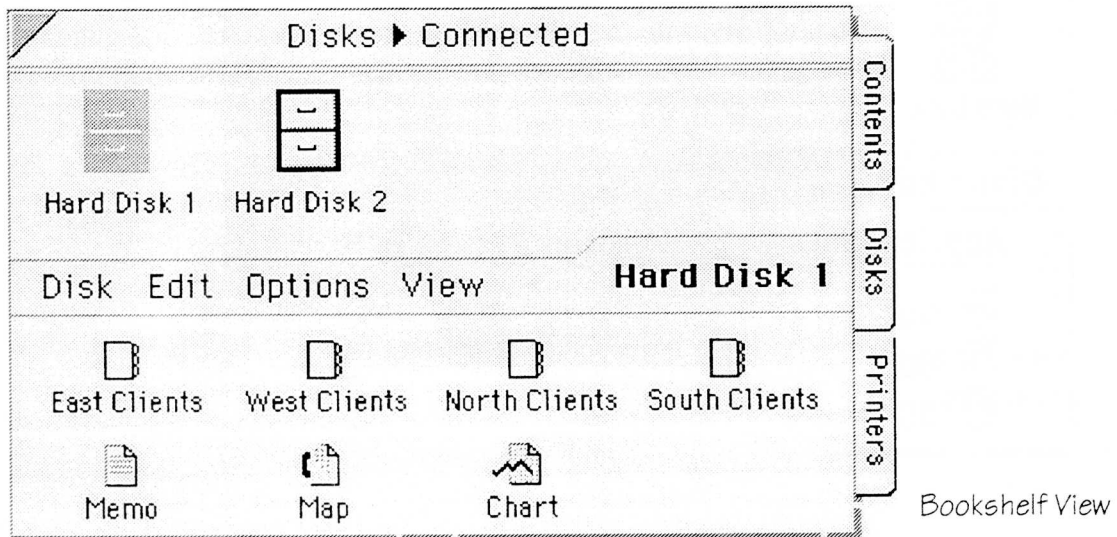
Directory View

The Directory view is for importing files and exporting documents. To import a file, the user simply drags it from a directory into a notebook. Conversely, to export a document, the user drags it from a notebook into a directory (see Chapter 7 for complete descriptions of the import and export operations).

The directory icon is the traditional file folder. The file icon is a plain rectangle with an arrow leading out of it, reflecting the fact that the file is in a foreign format and must be dragged into the PenPoint operating system to be used.

Bookshelf View

The figure below shows the Bookshelf view:



The per-disk bookshelves act just like the Bookshelf at the bottom of the notebook. They contain notebooks, documents, and accessories that the user can open and work with in the usual way.

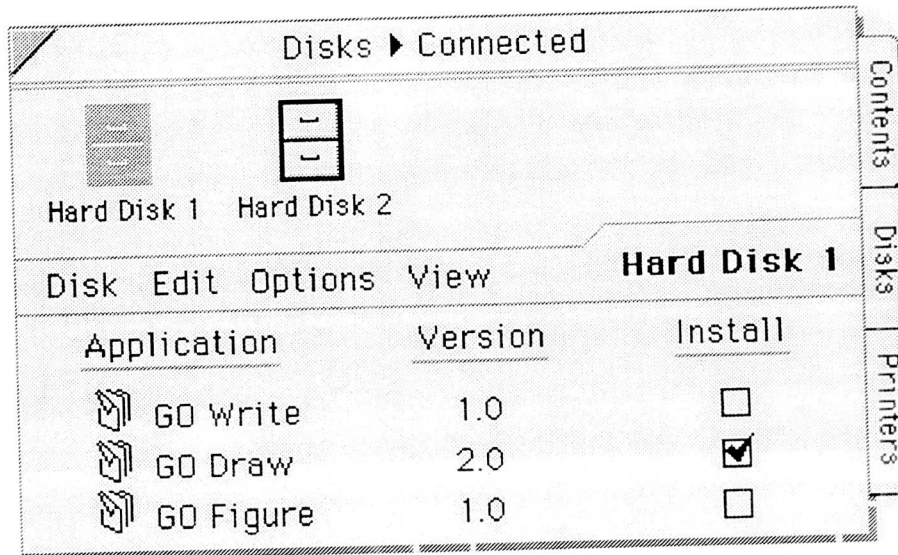
NOTE: The disk Bookshelf is a special directory on the disk that PenPoint manages; documents on the disk Bookshelf are not visible in the other views onto the disk.

Installable Software Views

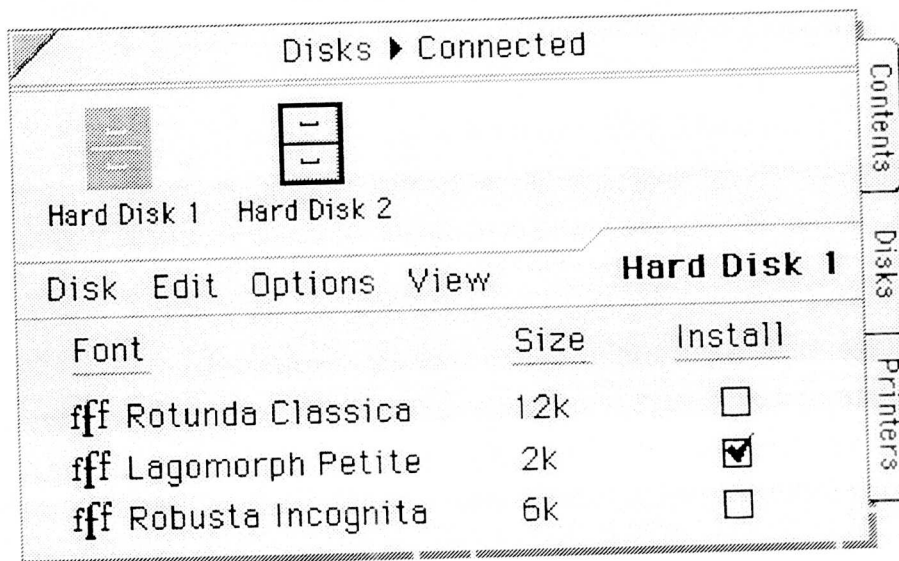
The installable software views allow the user to focus on any of the categories of installable software: Applications, Services, Handwriting, Gestures, Dictionaries, Fonts, and User Profiles.

Each installable resource has a checkbox that allows the user to install or deinstall the resource.

The figures below show the Applications view and the Fonts view:



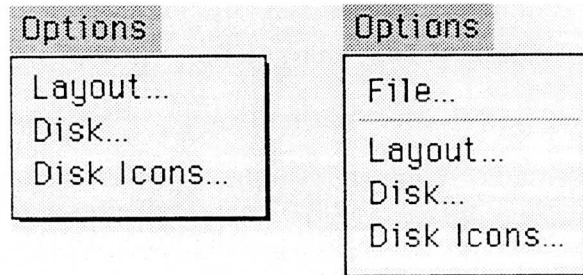
Applications View



Fonts View

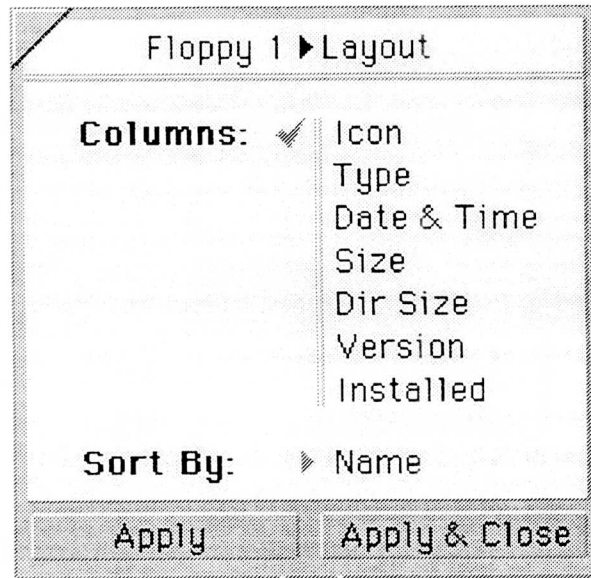
Option Sheets

Like all options menus, the **Options** menu for a connected disk changes dynamically with the selection. In the figure below, the menu on the left corresponds to the Applications view with nothing selected; the menu on the right corresponds to the Directory view with a file selected:



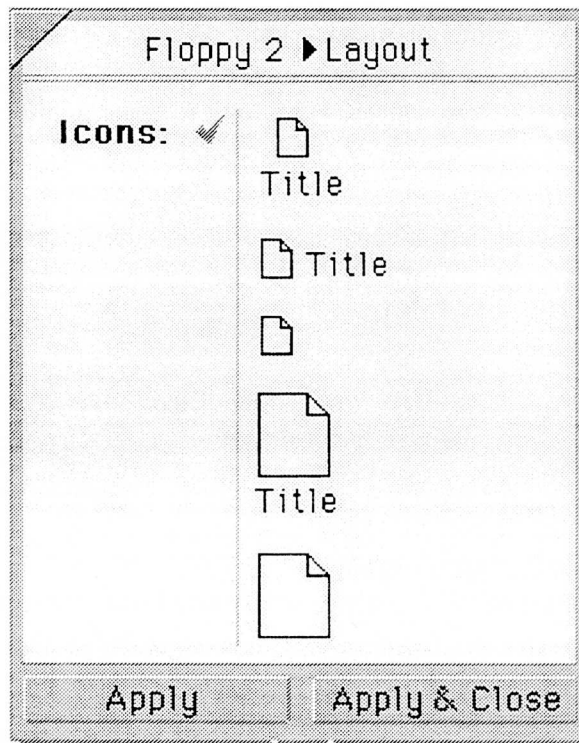
Options Menu for
Connected Disk

The **Layout** sheet also changes depending on the current view: directory, installable software, or bookshelf. For both directory and installable software views, the layout sheet contains the familiar PenPoint controls for displaying columns and sorting:



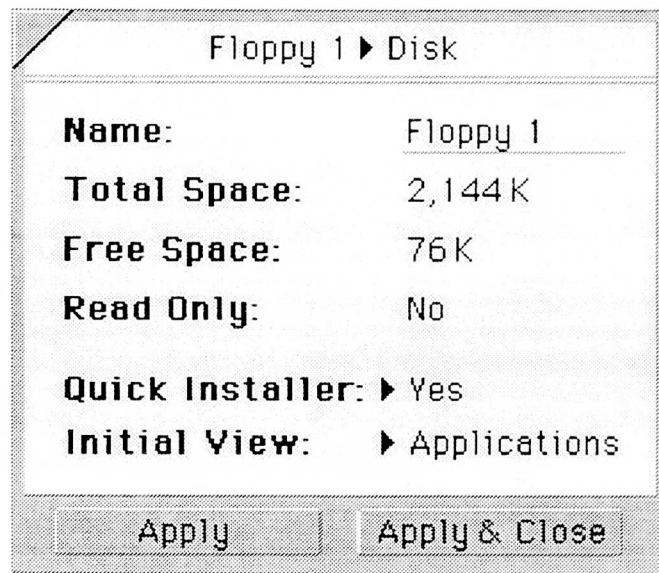
Layout Option
Sheet (Directory
View)

For the Bookshelf view, the **Layout** sheet contains the standard icon options:



Layout Option Sheet (Bookshelf View)

The figure below shows the **Disk** option sheet:



Disk Option Sheet

The bottom two controls pertain to the **Quick Installer**, described in the next section.

⚡ Quick Installer

As described earlier under Installable Software Views, the user can install software at any time from the Connected Disk page of the Connections notebook.

PenPoint provides a facility called the Quick Installer to streamline the installation process even further.

Any floppy disk can be marked as a “quick install” disk. When inserted into a disk drive, the quick-install disk automatically pops up an installation dialog:

Quick Installer for Floppy 1		
Applications on Floppy 1. Tap checkbox to install, or use the View menu to see other categories.		
Disk	Edit	Options View
Application		Version Installed
	GO Write	1.0 ☐
	GO Draw	2.0 ☒
	GO Figure	1.0 ☐

Quick Installer
Sheet

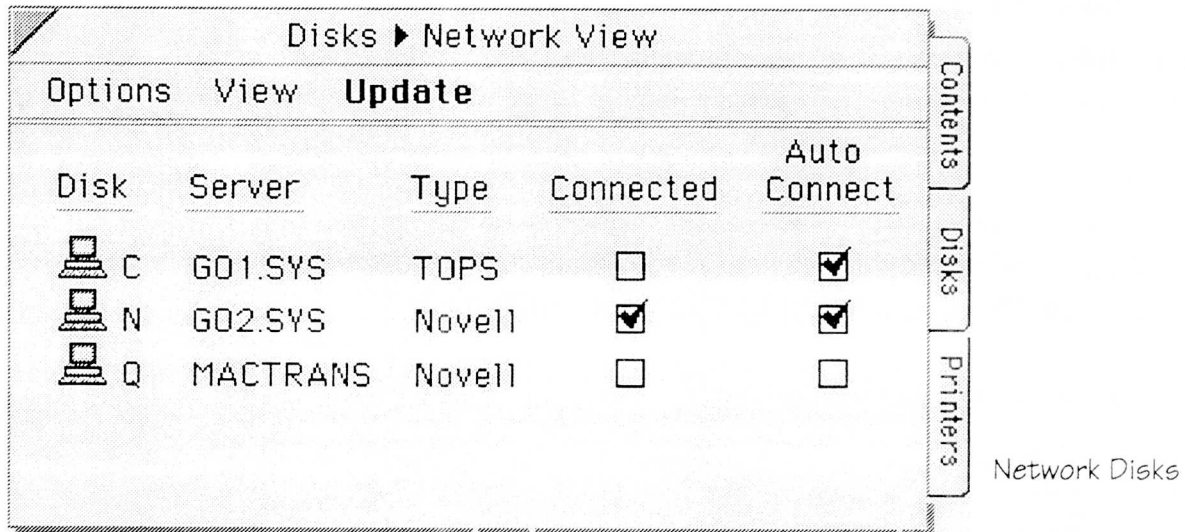
Software developers will typically deliver PenPoint applications on quick-install disks.

The user can control the quick-install behavior for any disk from its **Disk** option sheet. The two relevant controls are:

- ◆ **Quick Installer.** Allows the user to make any disk a quick-install disk.
- ◆ **Initial View.** Allows the user to specify which view comes up initially for the Quick Installer.

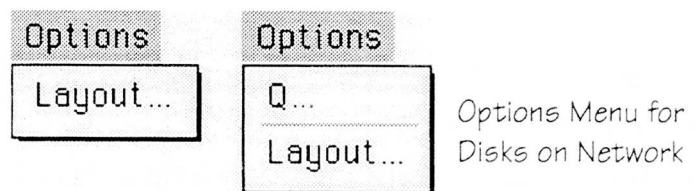
Network Disks

The Network View page of the Disks section shows the user the disks that are out on the network and can potentially be connected and accessed. The figure below shows a typical display:

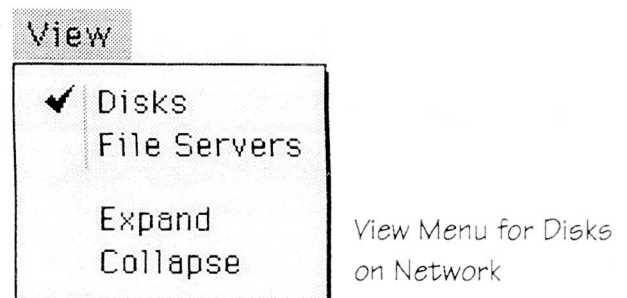


Menus

The Options menu has a Layout sheet and (when a disk is selected) a sheet for the disk:



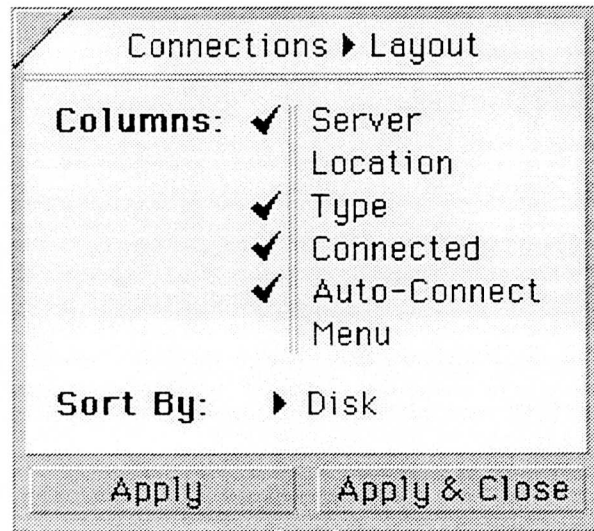
The View menu allows the user to view the network in different ways:



The Update button on the menu line refreshes the display to reflect the current state of the network.

Option Sheets

The figure below shows the **Layout** option sheet for the Network View page of the Disks section:



Layout Option
Sheet for Disks on
Network

The **Layout** sheet has the familiar options for displaying columns and sorting.

Checking the **Auto-Connect** checkbox for a disk causes the PenPoint™ operating system to automatically connect the disk whenever the appropriate network connection is made.

The **Menu** checkbox allows the user to specify that the disk appear on the menu that pops up from the check gesture in the disk area of the Connected Disks page.

Connected Printers

The Connected Printers page of the Connections notebook lists all of the printers that have been set up for use:

Printers ▶ Connected					
Edit Options Create Save					
Printer	Type	Model	Port	Enabled	
 Diconix	Dot Matrix	DicXPG	SerialA	☐	
 My Printer	LaserJet	HP 111D	Parallel	☒	
 Doc Printer	LaserJet	HP 111E	TOPS	☐	

Connected Printers
Page

The **Port** column tells whether the printer is connected directly through a port or indirectly over a network.

The **Save** button saves the selected printer to its home on disk (with no confirmation note). It is dimmed when there is no selection.

Menus

The figure below shows the Connected Printers menus:

Edit	Options	Create
Delete Rename...	Layout...	Dot Matrix... Laser Jet...

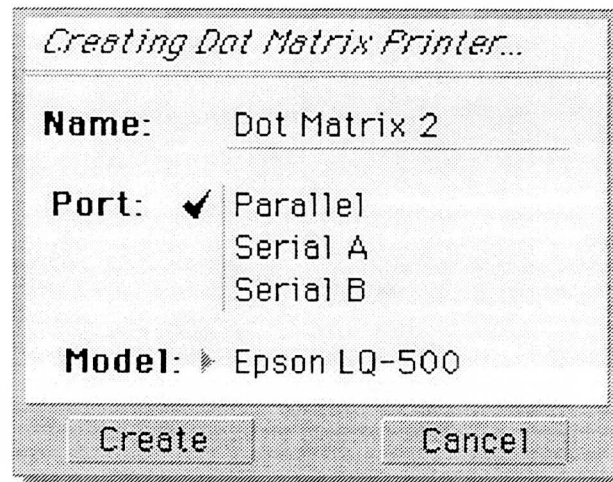
Connected Printers
Menus

When there is no selection, the **Options** menu has only the Layout sheet; when a printer is selected, the Printer sheet also appears:

Options	Options
Layout...	Doc Printer... Layout...

Options Menu for
Connected Printers

The **Create** menu contains all of the available printer types, as specified by the installed printer drivers. Choosing a printer type brings up the following dialog sheet:

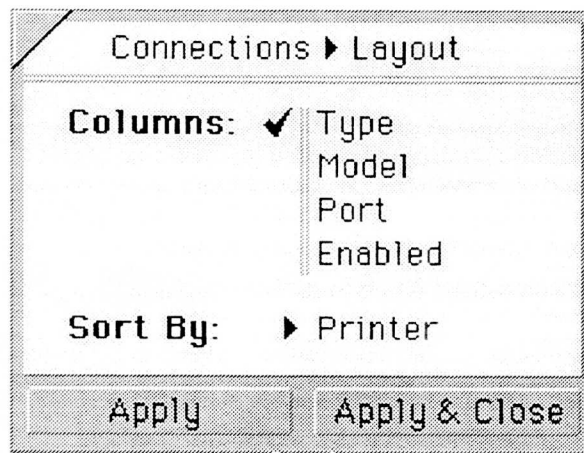


Printer Creation
Dialog Sheet

The **Model** list depends on the type of printer being created.

Option Sheets

The **Layout** sheet has the familiar controls for displaying columns and sorting:



Layout Option
Sheet

The figure below shows a typical printer option sheet:

Connections > Doc Printer

Model: Epson LQ-500

Port: ☒ Parallel
Serial A
Serial B

Paper Feed: ☒ Continuous
Hand Feed

Apply **Apply & Close**

Printer Option Sheet

All printer option sheets show the **Model** and have the **Port** control. The other controls depend on the type and model of the printer.

Network Printers

The Network View page in the Printers section is parallel to the Network View page in the Disks section:

Printers > Network View

Options View **Update**

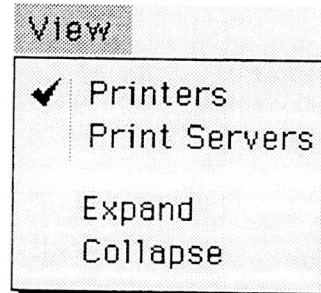
Printer	Type	Connected	Auto Connect
Mktg LaserWriter	LaserJet	☐	☒
Doc LaserWriter	LaserJet	☒	☒
Draft Printer	Dot Matrix	☐	☐

Contents Disks Printers

Printers on Network

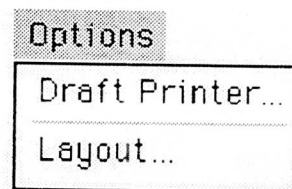
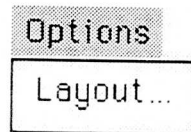
⚡ Menus

The **View** menu allows the user to view the network in different ways:



View Menu for
Printers on Network

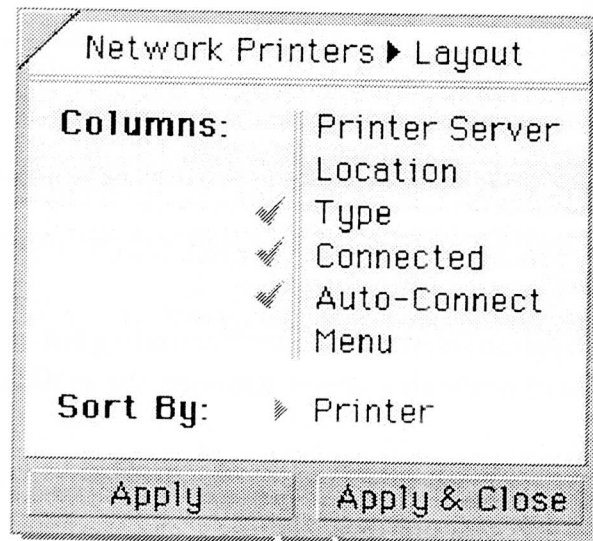
The **Options** menu has a **Layout** sheet and (when a printer is selected) a sheet for the printer:



Options Menu for
Printers on Network

⚡ Option Sheets

The **Layout** option sheet for the Network Printers page is parallel to the **Layout** sheet for the Network Disks page:



Layout Sheet for
Printers on Network

Chapter 14 / Stationery Notebook

The Stationery notebook is a special notebook containing stationery documents used as templates when the user creates new documents.

The figure below shows the table of contents for a typical Stationery notebook:

Stationery: Contents <1>		
Name	Page	Menu
 MiniText	2	
 MiniText	3	☒
 Business	4	☐
 Memo	5	☒
 MiniNote	6	
 MiniNote	7	☒
 Meeting Notes	8	☒
 ToDo List	9	☒

Stationery Table of Contents

The documents that have their **Menu** box checked appear on the system-wide **Create** menu described in Chapter 11.

When the user installs an application, the PenPoint™ operating system creates a section in the Stationery notebook with whatever stationery the application includes.

The user can create custom stationery at any time by simply moving or copying an existing document into the Stationery notebook.

Chapter 15 / In Box and Out Box

▼ In Box

The In box is where incoming email and fax documents are stored.

The In box takes the form of a notebook, with a section for each installed service that can receive documents or files.

The figure below shows a typical In box table of contents:

In Box: Contents <1>			
Document Edit Options View			
<u>Name</u>	<u>Page</u>	<u>Enabled</u>	<u>Status</u>
 PenMail	2	☒	Connected
 One Last Request	3		
 Chairs	4		
 Poker this Friday	5		
 Fax	6	☐	Disconnected

Typical In Box Table
of Contents

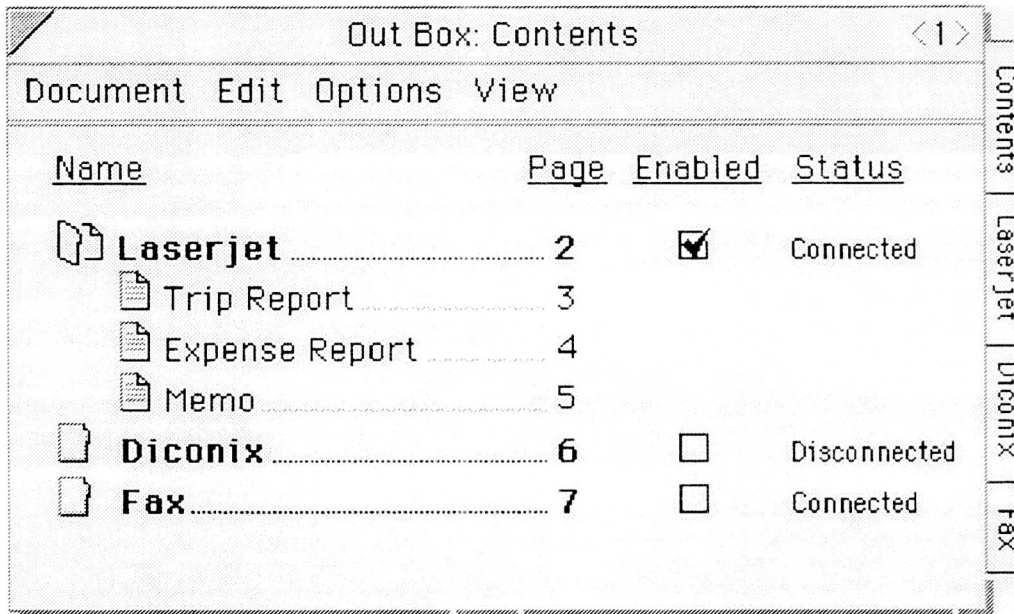
NOTE: In Release 1.0, the user cannot drag documents into the In box.

▼ **Out Box**

The Out box is where the PenPoint™ operating system stores outgoing documents before they are sent.

Like the In box, the Out box is a Notebook, with a section for each installed service that can send documents.

The figure below shows a typical Out box table of contents:



The screenshot shows a window titled "Out Box: Contents" with a "<1>" button in the top right corner. Below the title bar is a menu bar with "Document", "Edit", "Options", and "View". The main area contains a table with four columns: "Name", "Page", "Enabled", and "Status". The table lists documents grouped by service. On the right side of the window, there is a vertical sidebar with buttons for "Contents", "Laserjet", "Diconix", and "Fax".

<u>Name</u>	<u>Page</u>	<u>Enabled</u>	<u>Status</u>
 Laserjet	2	☒	Connected
 Trip Report	3		
 Expense Report	4		
 Memo	5		
 Diconix	6	☐	Disconnected
 Fax	7	☐	Connected

Typical Out Box
Table of Contents

The user can turn to the document in the Out box to view or edit it at any time before it is sent.

The menus are identical to the menus for the notebook table of contents, with the exception that the **Print** and **Send** commands are not present on the Out box **Document** menu.

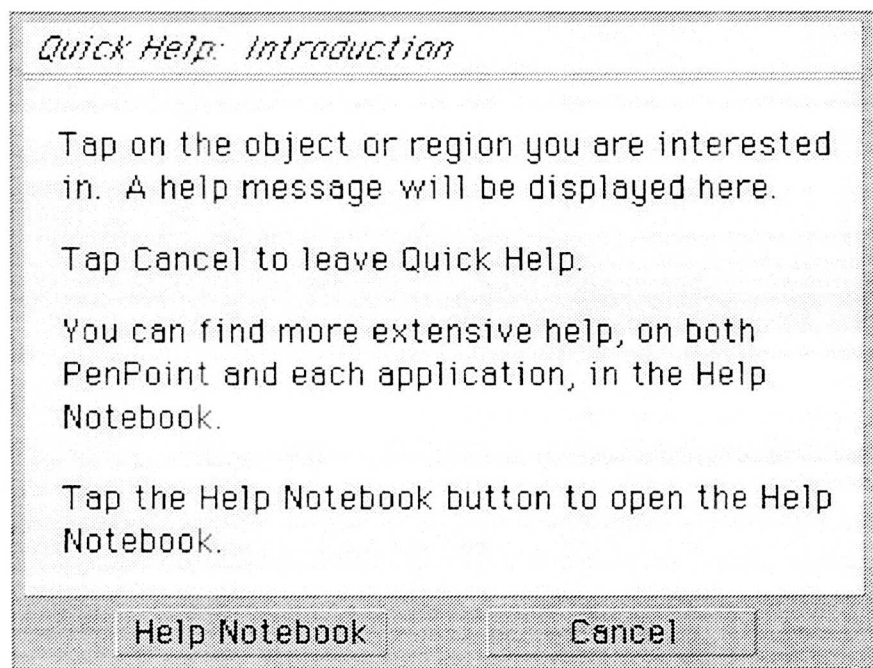
NOTE: As with the In box, in Release 1.0, the user cannot drag documents into the Out box.

Chapter 16 / Help

The PenPoint™ operating system provides two complementary help facilities: Quick Help, for getting brief, context-specific help for objects visible on the screen, and the Help notebook, for more extensive help on both PenPoint itself and installed applications.

Quick Help

Tapping the Help icon on the Bookshelf brings up the Quick Help sheet. Initially, the sheet contains instructions on how to use help:



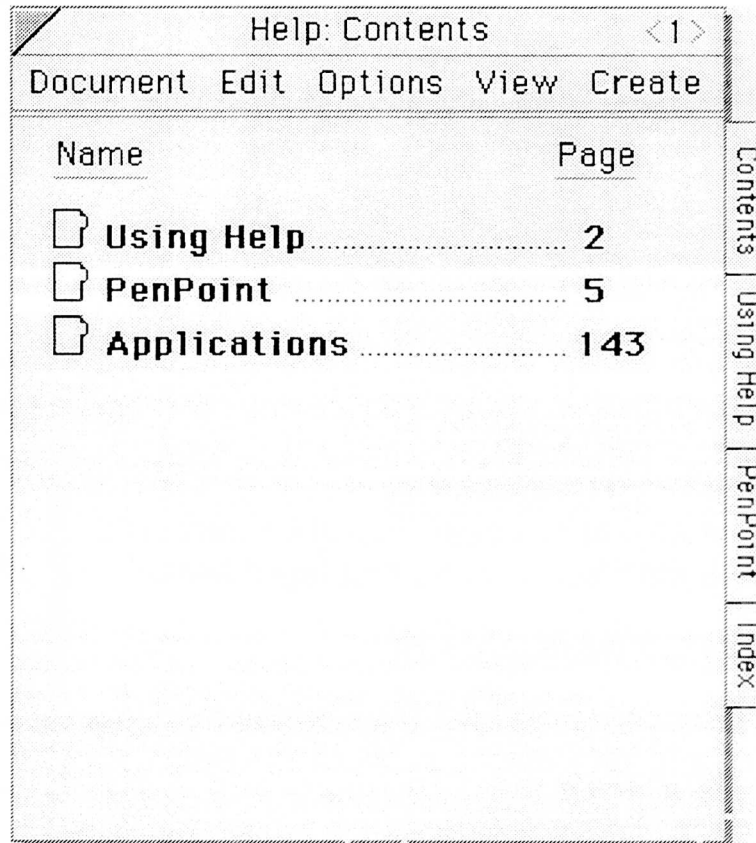
Quick Help Sheet

The Quick Help sheet is modal: As long as it is displayed, tapping on any object or region displays a brief help message for that object or region.

Tapping on a menu label (top level or submenu) displays the help message and pops up the menu as well. This allows the user to browse and get help on all the menu items.

Help Notebook

Double-tapping on the Help icon (or tapping the button on the Quick Help sheet) opens the Help notebook. The figure below shows the default table of contents:



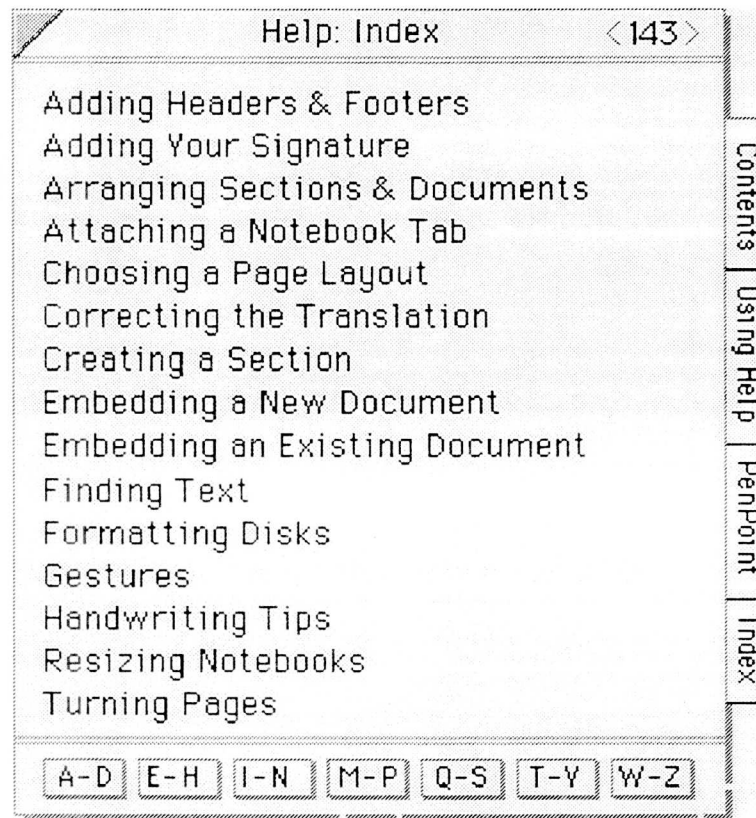
Help: Contents <1>	
Document Edit Options View Create	
Name	Page
 Using Help	2
 PenPoint	5
 Applications	143

Help Notebook Table
of Contents

The documents in the first two sections are provided by PenPoint; the documents in the **Applications** section are placed in the Help notebook by each application as part of the installation process.

While the PenPoint help documents are read-only, the user can add new documents to the Help notebook.

The Help notebook contains an index to the documents in the PenPoint section, as shown below:



Help Notebook Index

The index entries are implemented as reference buttons. The user can edit the index, adding, deleting, or renaming entries in the normal way.

The cork margin of the index contains reference buttons for quick scrolling to different parts of the index.

Part 5

Bundled Applications

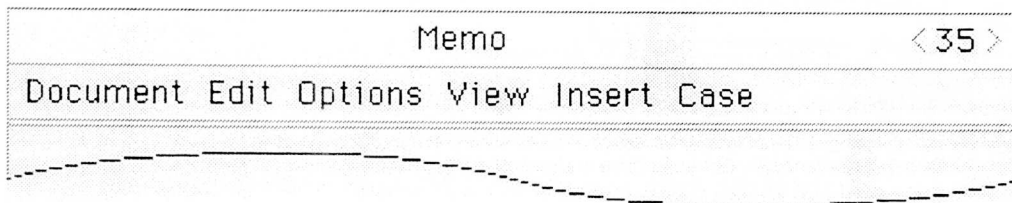
Chapter 17 / MiniText	129
Menus	129
Option Sheets	130
Gestures	133
Chapter 18 / MiniNote	135
Ink and Gesture Modes	135
Mode Toggle	135
Gesture Margin	135
Menus	136
Edit Menu	136
Arrange Menu	137
Pen Menu	137
Option Sheets	138
Gestures	140

Chapter 17 / MiniText

MiniText is a simple multi-font word processor. It is implemented as a “wrapper” around the PenPoint™ operating system text component.

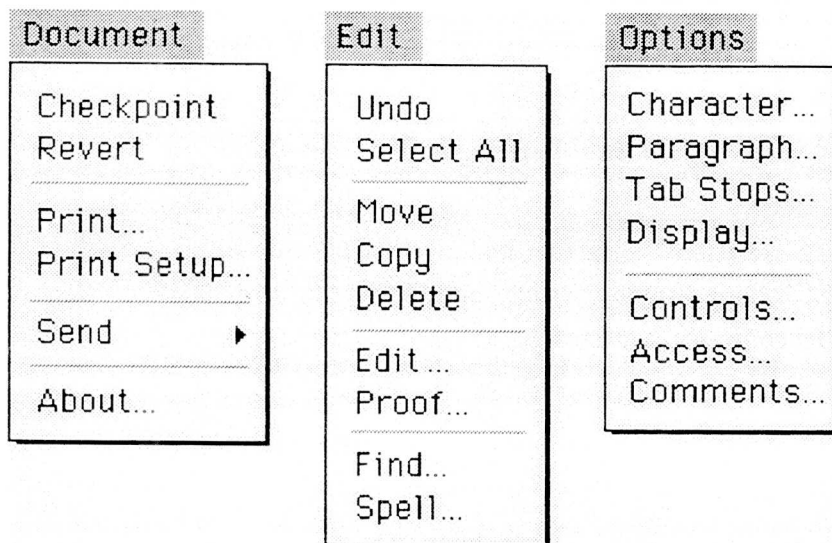
Menus

The MiniText menu line contains six menus:



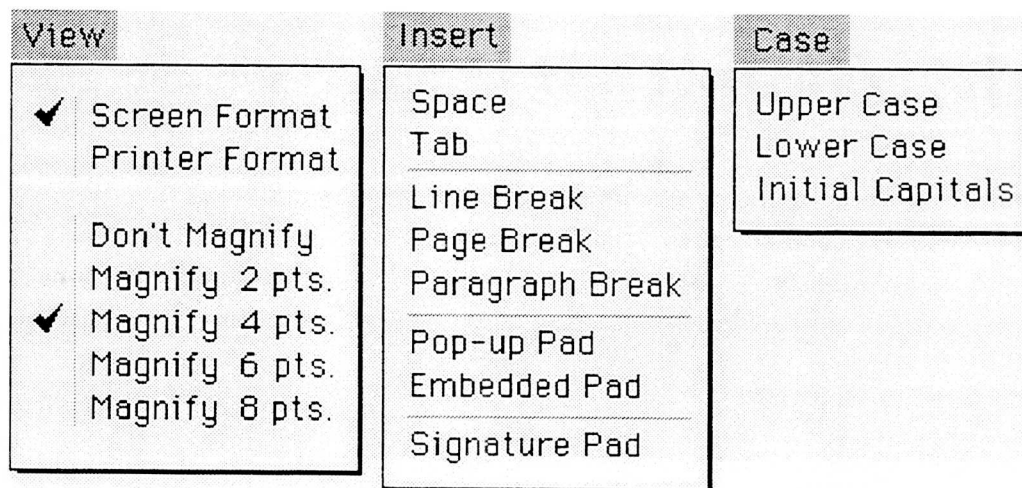
MiniText Menu Line

The contents of the three default menus are shown below:



MiniText Menus

The next figure shows the contents of the remaining three menus:



MiniText Menus

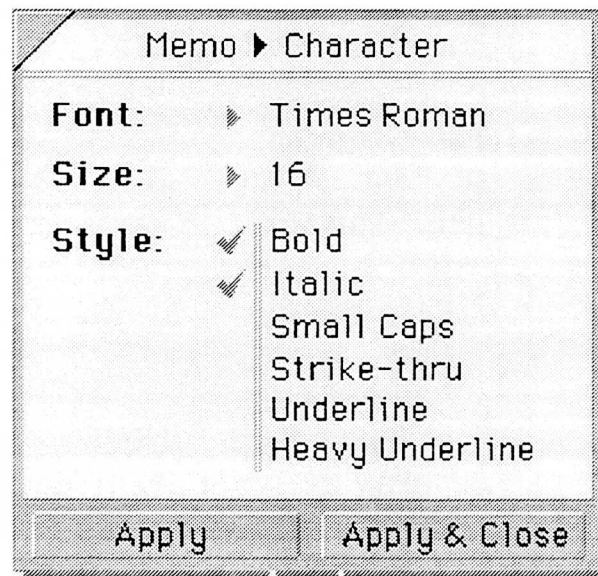
The **View** menu contains two checklists. The first allows the user to choose **Screen Format**, in which the text wraps at the right edge of the viewing region, or **Printer Format**, in which the text wraps at the right edge of the currently selected paper size (which is typically wider than the viewing region). The second list allows the user to specify a magnification factor that is applied only when in **Screen Format**.

▀ **Option Sheets**

MiniText has four option sheets: **Character**, **Paragraph**, **Tab Stops**, and **Display**.

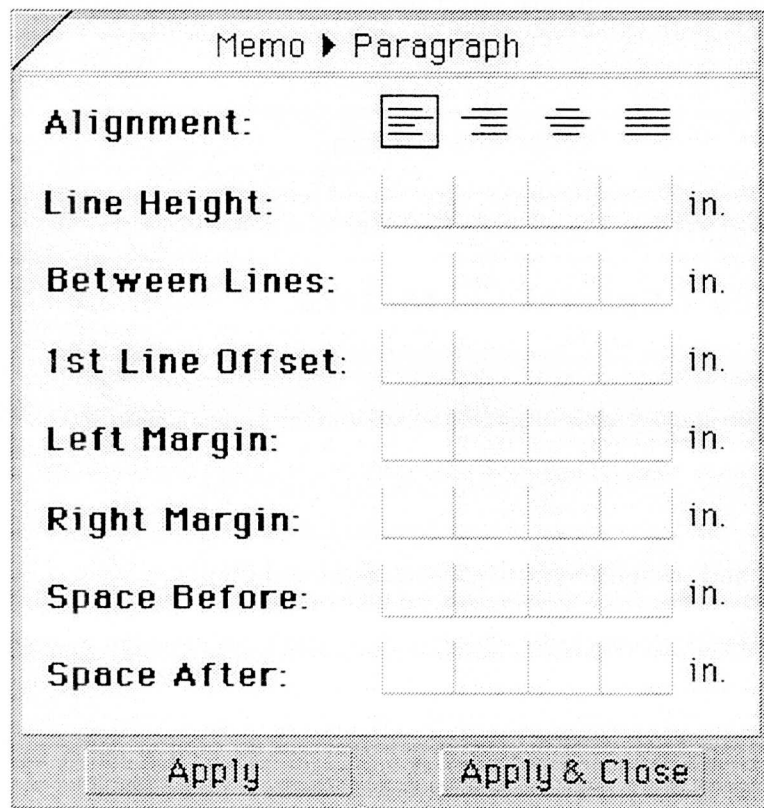
NOTE: The option sheets are actually implemented by the PenPoint text component that MiniText is built around, so the functionality they present is available anywhere the text component is used.

The Character sheet is for setting attributes of individual characters:



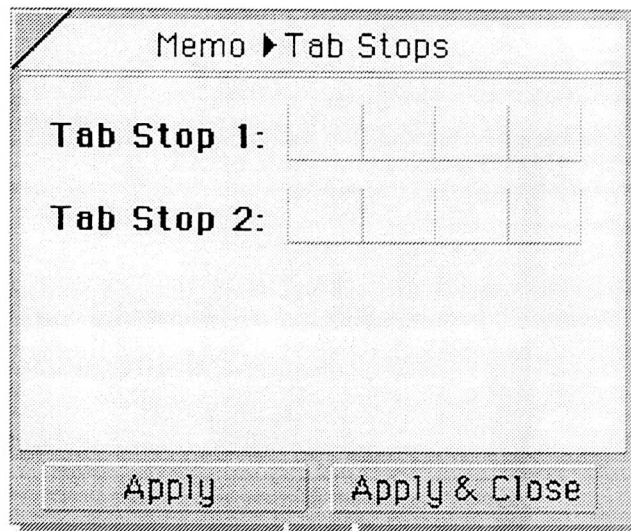
MiniText Character
Option Sheet

The Paragraph sheet is for setting attributes that apply to paragraphs:



MiniText Paragraph
Option Sheet

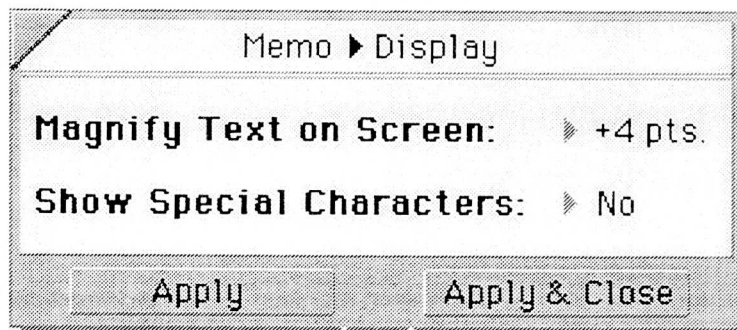
The **Tab Stops** sheet allows the user to create and modify tab stops:



MiniText Tab Stops
Option Sheet

The user can create a new tab stop with the caret gesture, and delete an existing one with the cross-out gesture.

The **Display** sheet contains two controls that affect the display of the document as a whole:



MiniText Display
Option Sheet

The **Magnify Text on Screen** control is functionally equivalent to the magnification checklist on the **View** menu. **Show Special Characters** toggles the display of tab, line break, and paragraph break characters on and off.

▼ Gestures

The PenPoint text component accepts all 11 of the PenPoint core gestures.

The following table lists all of the non-core gestures accepted in text.

Non-Core Gestures In Text

	<i>Double-Tap.</i> Selects a word.
	<i>Triple-Tap.</i> Selects a sentence.
	<i>Quadruple-Tap.</i> Selects a paragraph.
	<i>Double-Flick.</i> (Four directions.) Double-flick down scrolls to the beginning of the text, double-flick up scrolls to the end, double-flick left scrolls to the right edge, and double-flick right scrolls to the left edge.
	<i>Scratch Out.</i> Deletes any word touched by the gesture.
	<i>Circle-Line.</i> Brings up an empty editing pad to replace the word or selection.
	<i>Circle-Flick-Down.</i> Searches towards the end of the text for the next occurrence of the word (or selection) under the gesture.
	<i>Circle-Flick-Up.</i> Searches towards the beginning of the text for the next occurrence of the word (or selection) under the gesture.
	<i>Caret-Tap.</i> Creates an embedded insertion pad.
	<i>Up Arrow and Down Arrow.</i> Increases or decreases the point size for the word or selection by the increments in the character option sheet.
	<i>Up-Right.</i> Inserts a single character.
	<i>Down-Left.</i> Inserts a paragraph break.
	<i>Down-Left-Flick.</i> Inserts a line break.
	<i>Down-Right-Flick.</i> Inserts a tab.
	<i>Right-Up.</i> Capitalizes the first letter of the word (or of each word in the selection).
	<i>Right-Up-Flick.</i> Capitalizes the word or selection.

└	<i>Right-Down.</i> Makes the word or selection lower case.
B	Makes the word or selection bold.
F	Brings up the Find sheet, set to start the search from the point of the gesture.
I	Italicizes the word or selection.
P	Proofs the word.
N	Makes the word or selection “normal”—that is, turns off the bold, italic, and underlined attributes.
S	Begins Spell checking from the point of the gesture.
U	Underlines the word or selection.

Chapter 18 / MiniNote

MiniNote is a simple “ink processor.” It is implemented as a “wrapper” around PenPoint’s ink component.

► Ink and Gesture Modes

MiniNote has two input modes: ink mode and gesture mode.

In the default ink mode, pen input is simply accepted and stored as ink, with the following exceptions:

- ◆ Double-tapping over an ink object selects that object.
- ◆ The user can make gestures over selected objects, including circle to edit, cross-out to delete, press to move, tap-press to copy, flick to scroll, and so on.
- ◆ Tap-press over white space begins an area select.
- ◆ The left-right gesture anywhere toggles between the two modes.

In gesture mode, the user can make the usual core gestures—cross-out to delete, circle to edit, check for options, flick to scroll, and so on.

► Mode Toggle

At the right of the menu line is a toggle control for switching between input modes. The pen indicates that the document is in ink mode; the check indicates that it is in gesture mode:



Tapping on the pen switches to the check, and vice versa.

► Gesture Margin

MiniNote has a **gesture margin** that facilitates line-oriented operations.

The margin also provides additional mode feedback: It is grey in ink mode and white in gesture mode.

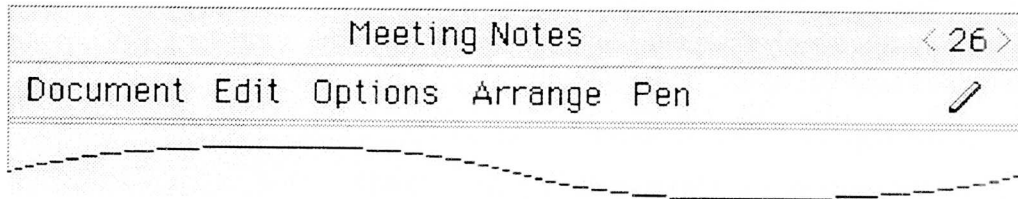
Most of the gestures that are accepted in the body of the document can be made in the gesture margin, where they are interpreted as applying to the entire line.

For example, double-tap selects the line, circle edits the line, down-left tidies the line, and tap-press selects the line.

Other useful margin gestures include right-down and right-up, to open and close white space by the amount of the vertical leg of the gesture. Left-down and left-up are also accepted, and are more easily made if the gesture margin is on the right.

▼ Menus

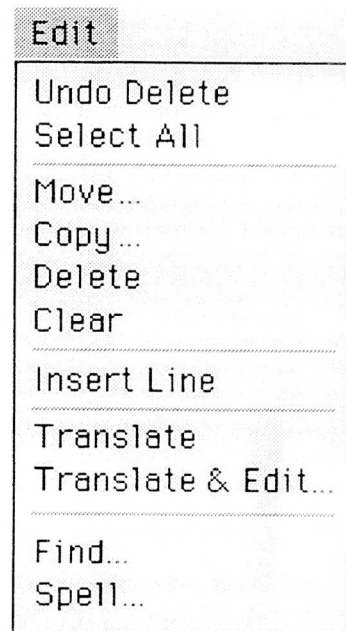
In addition to the mode toggle at the far right, the MiniNote menu line has five menus:



MiniNote Menu Line

▼ Edit Menu

The MiniNote **Edit** menu contains 11 commands:



MiniNote Edit Menu

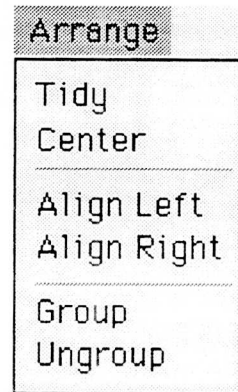
The **Delete** and **Clear** commands both delete the selected object or objects. The difference is that if all the objects on a line are selected, **Delete** deletes the objects and then closes the gap (by moving all the following lines up), while **Clear** leaves the empty line.

Insert Line inserts a new blank line above the current selection.

The **Translate & Edit** command translates the selected ink and pops up an edit pad containing the results of the translation. If the current selection is already translated, then the label simply reads **Edit**.

⚡ **Arrange Menu**

The **Arrange** menu contains commands to neaten up ink or text:



MiniNote Arrange
Menu

Tidy moves all the selected objects to the left as necessary to even up any gaps between objects.

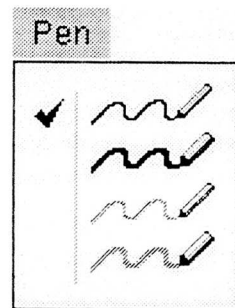
Center centers the selected objects.

The align commands operate on multiple lines. **Align Left** shifts the lines so they all start in the same column as the left-most line; **Align Right** shifts the lines so they all end in the same column as the right-most line.

Group joins all the selected objects into a single object; **ungroup** breaks all the selected objects apart into their constituent scribbles.

⚡ **Pen Menu**

The **Pen** menu allows the user to choose from a small set of pen widths and colors:



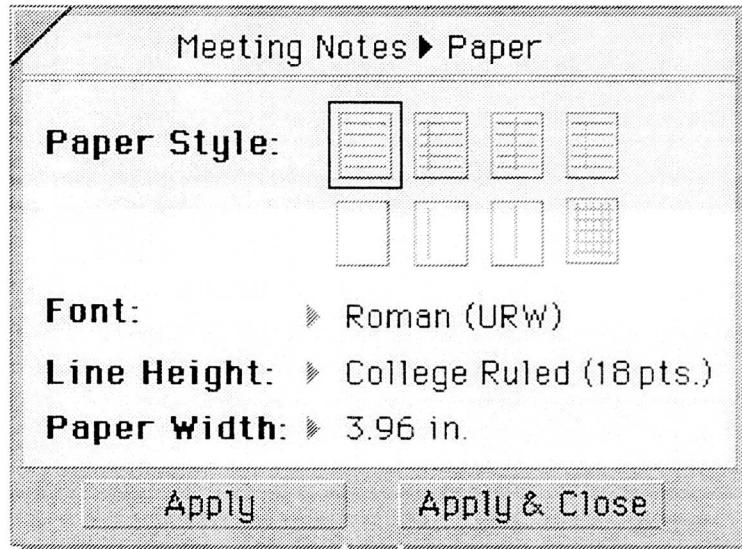
MiniNote Pen Menu

The **Pen** menu sets the style for any currently selected scribbles as well as for new objects.

Option Sheets

The PenPoint ink component around which MiniNote is built has two option sheets: **Paper** and **Pen**.

The **Paper** option sheet allows the user to specify the size and appearance of the paper:



MiniNote Paper
Options

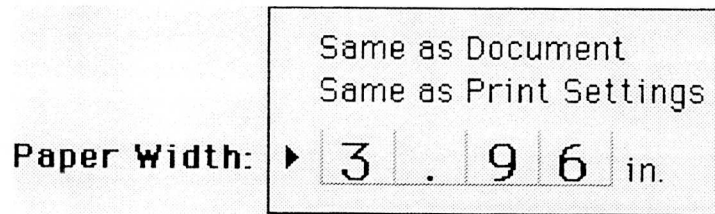
The **Font** control applies to all translated text in the document.

The **Line Height** menu has a text field for entering an arbitrary line height:



Line Height Menu

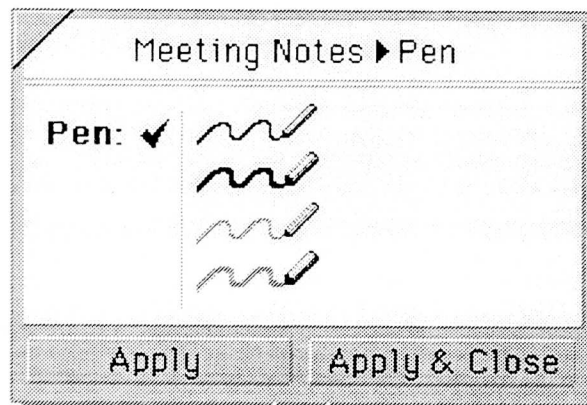
The **Paper Width** menu has an overwrite field containing the current paper width and two commands that affect the contents of the field:



Paper Width Menu

Choosing **Same as Document** has the effect of putting the current width of the viewing region into the overwrite field, so that a horizontal scroll margin is not needed. Choosing **Same as Print Settings** has the effect of putting the current paper width (taken from the **Paper Size** control on the **Print** sheet) into the overwrite field.

The **Pen** option sheet contains the same list of pen widths as the **Pen** menu:



MiniNote Pen
Options

▀ Gestures

The tables that follow describe all the gestures accepted by the PenPoint ink component.

Ink and Gesture Modes



Double-Tap. Selects a single object.



Plus. Toggles selected state.



Scratch Out. Deletes any ink or text objects touched by the gesture.



Tap-Press. Initiates an area select.



Flick Left-Right. Toggles between ink and gesture modes.

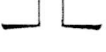


Caret-Tap. Pops up insertion menu with current date and time.



Pigtail. Deletes ink or text at the pen-down point.

Gesture Mode (or over selection in Ink Mode)

	<i>Tap.</i> Selects a single object.
	<i>Triple-Tap.</i> Selects line.
	<i>Plus.</i> Toggles selected state.
	<i>Press.</i> Initiates a move.
	<i>Tap-Press.</i> Initiates a copy.
	<i>Brackets.</i> Adjusts an existing selection.
	<i>Cross Out.</i> Deletes object (or selection).
	<i>Caret.</i> Pops up an insertion menu with current date and time.
	<i>Circle.</i> Edits text, translates and edits ink. All the selected ink is translated and appears in a pop-up edit pad.
	<i>Circle-Tap.</i> Translates ink to text without displaying an edit pad.
	<i>Check.</i> Displays the option sheet for the object (or selection).
	<i>Right-Down and Left-Down.</i> Opens white space as determined by the length of the vertical leg.
	<i>Right-Up and Left-Up.</i> Closes space as determined by the length of the vertical leg.
	<i>Flick.</i> (Four directions.) Scrolls to edge.
	<i>Double-Flick.</i> (Four directions.) Scrolls to beginning/end.
	<i>Circle-Line.</i> Brings up an empty edit pad for the word.
	<i>Undo.</i> Reverses the effect of the most recent operation.
	<i>Down-Left.</i> Tidies selected lines (evens out the space between all objects).
	<i>Down-Right.</i> Ungroups the selected object.

Gesture Mode (or over selection in Ink Mode) (Continued)

 *Up-Right.* Right-aligns the selected lines.

 *Up-Left.* Left-aligns the selected lines.

Capital Letter Accelerators

B Makes the word or selection bold.

F Brings up the Find sheet, set to start the search from the point of the gesture.

P Proofs a word.

S Begins Spell checking from the point of the gesture.

N Makes the word or selection “normal”—that is, turns off the bold attribute.

U Groups two adjacent scribbles. Start the gesture on one scribble, finish on the other.

Book II: Designing a PenPoint Application

Part 1 / Design Background

- Chapter 1 / UI Design Basics 149
- Chapter 2 / Designing for Mobile, Pen-Based Computing 151

Part 2 / Application Architecture

- Chapter 3 / Documents, Accessories, and Services 157
- Chapter 4 / Installation 162
- Chapter 5 / Managing Embedded Objects 166

Part 3 / User Interface Building Blocks

- Chapter 6 / Controls 175
- Chapter 7 / Menus 192
- Chapter 8 / Option Sheets and Dialog Sheets 200
- Chapter 9 / Status, Warning, and Error Feedback 210
- Chapter 10 / Help 215
- Chapter 11 / Designing Application Icons 216
- Chapter 12 / Putting the Building Blocks Together 223

Part 4 / Processing Pen Input

- Chapter 13 / Processing Gestures 231
- Chapter 14 / Processing Handwriting 241
- Chapter 15 / Presenting Input Modes 244
- Chapter 16 / Using Ink 254

Part 5 / Viewing and Manipulating Application Objects

- Chapter 17 / Scrolling and Zooming 263
- Chapter 18 / Scaling and Resizing 268
- Chapter 19 / Optimizing Layout for Small Screens 274
- Chapter 20 / Selecting 278
- Chapter 21 / Moving and Copying 285
- Appendix A / PenPoint Design Checklist 295

Part 1

Design Background

DESIGNING A PENPOINT APPLICATION

PART 1 / DESIGN BACKGROUND

■ Chapter 1 / UI Design Basics	149
User-Centered Design	149
Consistency	149
Graphic Design Fundamentals	149
Wording Fundamentals	150
Direct Manipulation	150
Iterative Development	150
■ Chapter 2 / Designing for Mobile, Pen-Based Computing	151
Designing for the Notebook Metaphor	151
Designing for the Pen	152
Designing for the Embedded	152
Designing for Scalability	153
A New Level of Consistency	154
A Design Challenge for the '90s	154

Chapter 1 / UI Design Basics

The PenPoint™ operating system's user interface belongs in the tradition of Graphical User Interfaces (GUIs) that stem from the pioneering work on the Xerox Alto and Star in the 1970s. During the past decade, GUIs have become the standard way of interacting with personal computers.

As GUIs have passed into general usage, a significant body of experience has been amassed, both from formal research in human factors and from real-world experience with many environments and applications. This body of experience amounts to a design tradition.

This chapter mentions some of the accepted tenets that are part of this tradition. Each of these, and others, have been written extensively about in textbooks, design guides, articles, and research papers. If you have developed applications for other GUIs, you will no doubt be familiar with these principles in one form or another from your own experience.

These design basics are mentioned here to provide a foundation for the pen-centric perspective offered in Chapter 2.

▼ User-Centered Design

The user should direct the computer, not the other way around—let the user drive.

▼ Consistency

The user typically uses several applications, each of which may come from a different vendor, and each of which has many features. The only way the user will be able to cope with such a rich world is if the different aspects of the environment behave consistently. Conventions for both the look and feel of the environment are crucial.

▼ Graphic Design Fundamentals

The most basic visual principles are balance, proportion, and the maintenance of a clear distinction between foreground and background. Practices from graphic design that aid in achieving balance, proportion, and readability include laying items out on a grid, and proper use of white space.

▼ **Wording Fundamentals**

Fundamentals include consistent capitalization, use of short, clear words, clear wording of messages, and the use of terms that are from the user's world (not the world of the programmer or hardware technician). Manners matter, too; don't be condescending or rude to the user.

▼ **Direct Manipulation**

Direct manipulation is the illusion of acting on a "thing" represented on the computer screen. The thing might be text, the contents of a cell in a table, a shape in a drawing, a document, and so on.

In mouse-based systems, direct manipulation typically means dragging objects. The simple act of dragging is very powerful, since it can be interpreted as performing different operations depending on the context: moving, creating, destroying, printing, sending, and so on.

PenPoint adds other types of interaction that have the same immediacy as dragging: flicking over a document to scroll more information into view, making a cross-out to delete a paragraph, circling the document's title to edit it, and so on.

▼ **Iterative Development**

The only way to ensure that your customers will be satisfied with your application is to design, prototype, get user feedback, then redesign, modify the prototype, get more feedback, and so on. Repeat until users report satisfaction.

This is a laborious process, but it works. To paraphrase what Churchill said about democracy...it's a lousy system, but has the one virtue of being better than all the others.

Chapter 2 / Designing for Mobile, Pen-Based Computing

The principles summarized in Chapter 1 provide a foundation for the design of GUI applications. But they are not sufficient. Designing a mobile, pen-based application is not the same as designing a desk-bound, mouse-based application.

The key innovations of the PenPoint™ operating system—including the notebook metaphor, gestures, the Embedded Document Architecture, and scalability—point to a different way of thinking about computers, and a different way of thinking about application design.

The sections that follow suggest some of the implications of these features for application design.

▼ Designing for the Notebook Metaphor

The paper-like world suggested by the notebook metaphor is rich in design possibilities. We are all familiar with notebooks, forms, sheets, pads, tabs, bookmarks, sticky notes, and so on. The graphical conventions used in this world are equally rich: checkboxes, borders, margins, grids, and rules of every stripe.

As far as possible, all the elements of the PenPoint user interface have been designed to make visual and conceptual sense in the context of this paper-like world. By designing within the notebook metaphor, you will help make the overall PenPoint environment visually and conceptually coherent.

In particular, unless you are deliberately emulating a calculator, stereo panel, or some other type of control panel, avoid the “gadget” look, with raised buttons and switches, etched lines, heavy shadows, and other visual embellishments.

Of course, the notebook metaphor can only be carried so far. For example, buttons are an important element of the PenPoint user interface, and no one ever tapped a button on a paper document to make something happen.

But the magic of a powerful metaphor lies not in its precision, but in its ability to bring together different worlds in a useful way. The PenPoint user interface brings together the world of paper, pencils, notebooks, and writing with the world of computers, with its buttons, icons, links, and automatic commands.

▼ Designing for the Pen

The pen is clearly a different sort of input device than the mouse and keyboard combination.

- ◆ The pen is a single, simple device that allows the user to both point at objects on the screen and enter data.
- ◆ Where the mouse requires the user to first select an object, then choose the command, the pen allows the user to specify both the operand and the operation with a single gesture.
- ◆ Where the keyboard requires an insertion point, and some sort of feedback to mark it, the pen allows the user to write wherever desired, or to make a gesture to indicate where to insert text.
- ◆ Where the mouse is a relative input device, requiring a pointer on the screen as its surrogate, the pen is an absolute device that does not require a pointer.

All of these differences, and more, need to be taken into account when designing a pen-based application.

But over and above the specific characteristics of the pen as an input device, the real challenge is to design applications that fully exploit the pen. That requires stepping back and rethinking traditional approaches.

▼ Designing for the Embedded Document Architecture

The PenPoint Application Framework implements embedding and reference buttons transparently for all applications, so you don't have to write special code to implement the basic functionality.

There are some things you should do to take full advantage of these features. For example, most applications will want to be aware of embedded objects for layout purposes. And when the user taps a reference button that is linked to an object within your application, the application should scroll the object into view and select it.

But PenPoint's Embedded Document Architecture (EDA) has implications beyond the details of handling embedded objects. It makes possible a modular approach to application design and development, which benefits both the application vendor and the user.

Current PC environments encourage a "monolithic" model of application architecture in which all applications tend to converge: A word processor adds support for drawings, tables, and charts; a spreadsheet adds support for word processing and charts; and so on. This trend has unfortunate consequences for both the developer and the user. Development and testing cycles are stretched

out, and the user is locked into using all the functionality bundled into the monolithic application, like it or not.

With PenPoint's EDA, instead of writing a complex, monolithic application that tries to be all things to all people, you can think in terms of a suite of simpler, more focused applications, knowing that the *user* can combine them as needed. And you can develop and test each unit of functionality separately.

When designing your application, then, one of the first things to ask is, "How can I divide this application up into modules of functionality that the user can combine as needed?"

Also, it may pay to investigate whether there are other applications or components, provided either by GO or by other PenPoint application vendors, that can provide some part of the functionality you need.

▼ Designing for Scalability

Scalability is a term that encompasses many variables in the way information is presented.

The most basic variable is the number of pixels on the display. Traditional GUIs assume a full-sized desktop display, typically with at least 640 x 400 pixels. PenPoint computers, however, will come in a wide variety of form factors. Of particular importance are small, pocket-sized devices. The minimum size for a PenPoint display is 320 x 200 pixels.

Eventually, the small, mobile devices will be complemented with stationary, wallboard-sized work surfaces to facilitate people working together in groups.

There are individual differences to be taken into account. Some users have good eyesight, some poor. Some users prefer to see more information at the expense of readability, others prefer to see less and have it be more readable.

Another variable is the setting in which the computer is being used. Unlike deskbound computers, which are used under controlled lighting conditions, mobile computers are used under widely varying conditions. For instance, someone who normally reads menu text at 10 points may prefer a much larger font when trying to read the computer display in a moving vehicle, or walking from a bright room to a dimly lit one.

To support the user in the face of this diversity, the PenPoint operating system allows the user to change the layout of the system at any time, specifying portrait or landscape orientation, left-handed or right-handed layout, and different font sizes and type faces.

Taken together, these factors mean that the display parameters that both the system and the application need to take into account vary greatly.

The PenPoint User Interface Toolkit includes a powerful **automatic layout** facility that ensures device independence and scalability. The standard components of the PenPoint user interface make use of this facility, so any menus, option sheets, and notes you use in your application are rendered in a device-independent and scalable way.

But this doesn't solve all the scalability-related presentation issues. As an application designer, you need to understand the layout-related issues so that your application behaves well under optimum circumstances and adequately under more restricted conditions.

▼ **A New Level of Consistency**

The wide acceptance of GUIs during the '80s has shown the value of consistency in user interface design. As pen-based computers extend the use of computers to new groups of people—people who are less technical and have less time to spend learning computer arcana—consistency will become even more important.

The PenPoint operating system lays the foundation to support this new class of users by providing consistent interfaces for many commands and options, including some that in traditional environments are presented differently by each application.

Such commands include **Print** and **Print Setup**, **Send**, **Find**, **Spell**, **Import** and **Export**, and many others. Of course, you can customize these interfaces if you need to. But in most cases using the standard building block that PenPoint provides will allow you to concentrate on bringing the particular added value of your application to the marketplace.

In addition, PenPoint supports a set of 11 **core gestures** that all applications should implement where appropriate.

Another example of how PenPoint promotes consistency is the **drag & drop** method of moving and copying objects, which allows the user to move or copy objects of all types—documents, icons, text, figures in a drawing program, and so on—in the same way.

▼ **A Design Challenge for the '90s**

This chapter has briefly sketched out some of the new design territory that the PenPoint operating system opens up.

The rest of this book is a guidebook for exploring that territory. It spells out in detail how to design your application to take full advantage of PenPoint's features—the notebook metaphor, gestures and handwriting recognition, embedding and reference buttons, mobile connectivity, scalability, and consistency.

Part 2

Application Architecture

DESIGNING A PENPOINT APPLICATION

PART 2 / APPLICATION ARCHITECTURE

▼ Chapter 3 / Documents, Accessories, and Services	157
Documents	157
One Document per Notebook Page	157
Multi-Page Documents	157
Non-Document Applications	159
Transfer Services	160
▼ Chapter 4 / Installation	162
Document Stationery	162
Accessories	163
Services	163
Flexibility versus Simplicity	165
▼ Chapter 5 / Managing Embedded Objects	166
Basic Support for Embedded Objects	166
Storing Embedded Objects	166
Displaying Embedded Objects	166
Intelligent Layout of Embedded Objects	168
Special Handling of Open Documents	168
When Opening In-Line Is Not Appropriate	169
Marking Locations within a Document	171
Reference Buttons	171

Chapter 3 / Documents, Accessories, and Services

One of the first decisions you need to make is how to present your application in relation to the PenPoint notebook.

The most common approach is to present your application as a document in the PenPoint notebook. PenPoint applications are not limited to the document format, however. Other standard ways of presenting applications include accessories that pop up over the notebook, separate notebooks, and services that work with the In box and Out box.

▼ Documents

If your application makes sense as something that could be found in a notebook, present it in the form of a document.

Under the PenPoint™ operating system, this category is intended to be broad, including not only such familiar documents as word processors, drawings, forms, and spreadsheets, but applications that would not ordinarily be thought of as documents, such as database front ends, games, and so on.

▼ One Document per Notebook Page

The basic user model for the PenPoint notebook is that each page of the notebook contains a single document. The document itself may contain more information than can be viewed at one time; the user can scroll the offscreen portions of the document into view by using the scroll margins or the flick gestures.

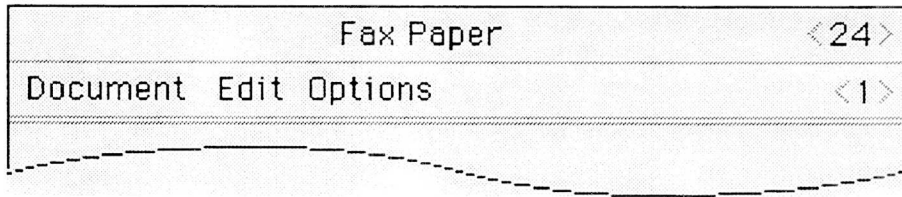
▼ Multi-Page Documents

Some applications are strongly oriented towards the printed page. Word processors, for example, are designed to produce documents that can be viewed on the screen within a variable-sized region, but are correctly formatted onto pages when printed. In such a document, the text can flow from page to page as text is added and deleted.

Other types of documents, such as forms, are even more strongly page-oriented. In a form, the layout of each page is fixed ahead of time, and information does not flow between pages. If your application is page-oriented, you should provide a control for paging within the document, as described in the following two sections.

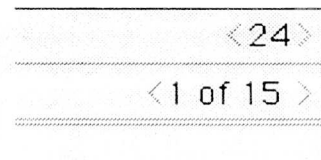
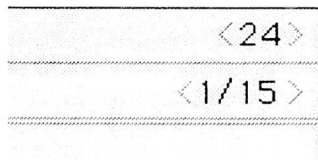
❖❖ **Local Page Numbers**

If your document has numbered pages, put the page control at the right of the menu line, directly below the notebook page number:



Local Page Number at
Right of Menu Line

The page control provided by the PenPoint User Interface Toolkit gives you the option of showing the total number of subpages. Two formats are supported:

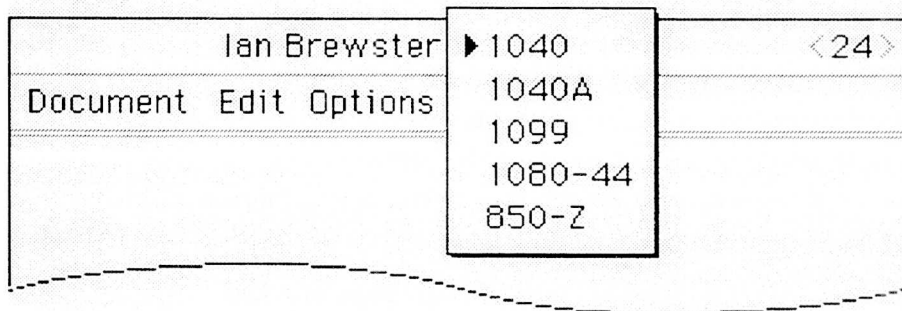
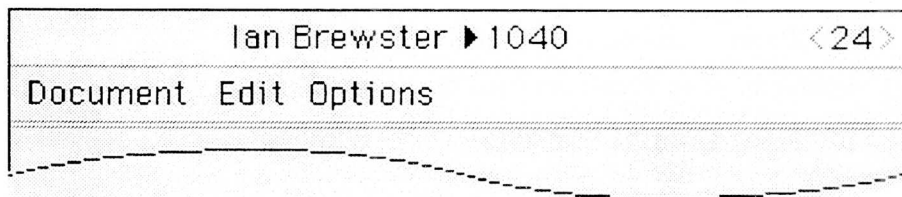


Page Number Controls
with Page Count

You can also specify that a menu of available pages be displayed when the user taps the page number.

❖❖ **Pop-Up List on Title Line**

If your application has a limited number of named pages, consider using a pop-up list on the title line, as shown below:



Pop-Up List on
Document Title Line

Non-Document Applications

The document format is appropriate for most applications, but not all. The following table describes the other categories of PenPoint applications and when to use them.

Types of PenPoint Applications

Type	When to Use	Examples
Pop-Up Accessory	Present your application as an accessory when it doesn't make sense, conceptually or visually, to present it as a document in the notebook. Anything that makes more sense as a three-dimensional control panel (rather than a two-dimensional piece of paper) is a good candidate for an accessory.	Clock. Calculator. Control Panel for an information service.
Notebook	If your application consists primarily of a lot of information, then it may be appropriate to package it in the form of a separate notebook. The Help notebook is an example of an application that is structured as a notebook. You can either use the standard PenPoint notebook, or you can tailor the notebook by subclassing it.	Help notebook. Service manual. Collection of maps of a city or region. Book reader.
Transfer Service	Transfer services manage the sending of information through the Out box, or the receiving of information through the In box, or both.	Printer driver. Fax. Email.
Background Service	Background services are typically called by other applications to provide some kind of information.	Database server. Information service.

Transfer Services

This section gives examples of services that use the Out box to send documents. The material applies as well to services that use the In box to receive documents.

The figure below shows a typical Out box table of contents:

Name	Page	Enabled	Status
Laserjet	2	☒	Connected
Trip Report	3		
Expense Report	4		
Memo	5		
Diconix	6	☐	Disconnected
Fax	7	☐	Connected

Typical Out Box Table of Contents

Note that each service has its own Out box section.

Transfer services are either **connected** or **disconnected** at any given time, and should automatically detect changes in state. (Some hardware may not support automatic detection of connection and disconnection. In such cases, the service should always report that it is connected.)

When a transfer service becomes connected, it should begin processing the information in its queue.

The checkboxes in the **Enabled** column allow the user to override this automatic behavior. If the checkbox is not checked, then the service should not process information, even if it is currently connected.

The **Status** column is for brief status messages. When the service is idle, the message should read either "Connected" or "Disconnected," as appropriate. When an operation is in progress, the service should show a message such as "Printing page 1...."

By default, the table of contents for the service shows a simple list of the objects that are queued up. The figure below shows a typical table of contents for a printer service:

Laserjet <2>	
Document Edit Options View	
Name	Page
 Trip Report	3
 Expense Report	4
 Memo	5

Default Out Box Table of Contents for a Printer Service

By default, user actions in the service table of contents are limited to deleting or re-ordering entries. Your service can modify its table of contents to present additional information or options.

The example below shows a customized table of contents for a hypothetical fax service:

PenFax <2>	
Document Edit Options ☒ Enabled	
Sending "Practice Notes"; page 2...	
Document	To
 Practice Notes	Chris Gatling
 Map to Arena	Victor Alexander
 Stats	Billy Owens
	Send Time
	Immediate
	5:00 P.M.
	12:00 A.M.

Customized Service Table of Contents in Out Box

The service has added a control at the right of the menu line to allow the user to disable the service, and added a message line below the menu line. Instead of page numbers, there is a **To** column showing the addressees, and a **Send Time** column showing when each document is scheduled to be sent. The times are editable (via the circle gesture) so that the user can turn to the fax service section at any time and change the send time.

Chapter 4 / Installation

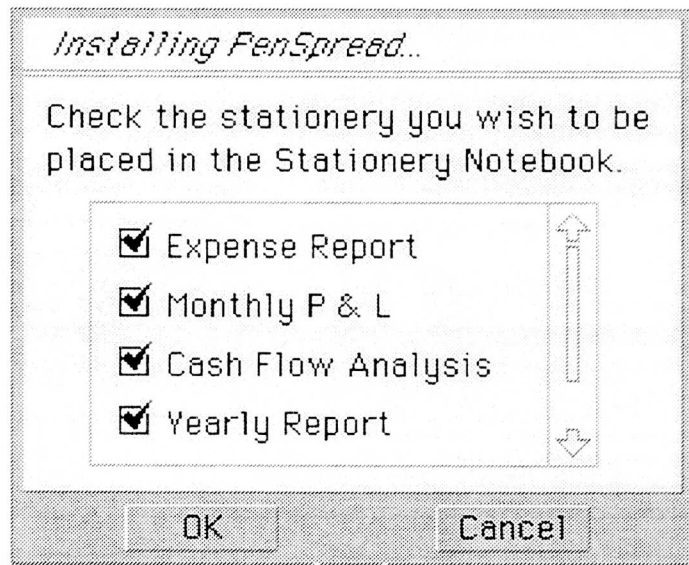
For applications that are structured as documents, the installation process results in stationery being placed in the Stationery notebook. If your application is *not* document-oriented, display a note at the end of the installation process telling the user where the application has been installed and how to access it. You can also display dialog sheets or notes at installation time to allow the user to choose additional stationery, set configuration options, or point the user toward essential learning material such as a "Read Me First" document.

Document Stationery

If your application is document-oriented, you should take advantage of the Stationery notebook to provide preconfigured stationery documents to your customers.

As part of the installation process, you can specify that PenPoint create a section in the Stationery notebook named for your application, and place one or more documents in it. The usual practice is to create at least one stationery document, named for the application.

Even better than installing a predetermined list of stationery automatically, consider allowing the user to choose from a list:



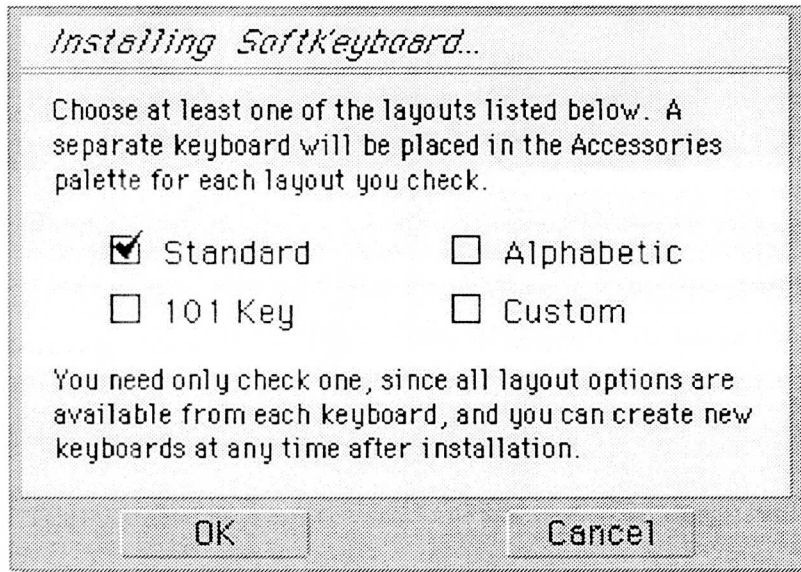
Dialog Sheet for
Installable Stationery

Generate the list of possible stationery from the contents of the special stationery directory on the installation disk.

▼ Accessories

If your application is an accessory, the installation process adds the application's icon to the Accessories palette.

As with stationery, you can specify multiple accessory instances for a given application, either with or without user intervention. The figure below shows a dialog sheet for a keyboard accessory that supports multiple layouts.

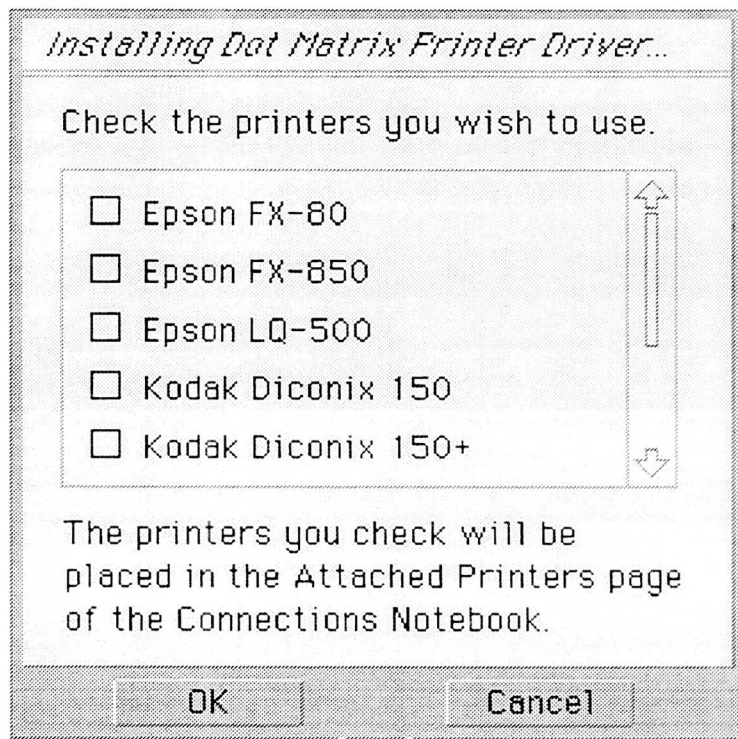


Dialog Sheet for
Installable Accessory
Instances

▼ Services

A service may have its own analog to stationery. For example, a printer driver for a particular class of printers typically supports many makes and models of printers. The user can add a printer to the list of available printers at any time by turning to the Attached Printers page of the Connections notebook and making the caret gesture to create a new printer.

As a convenience, the service should present the list of supported printers to the user during installation:



Installation
Configuration Dialog

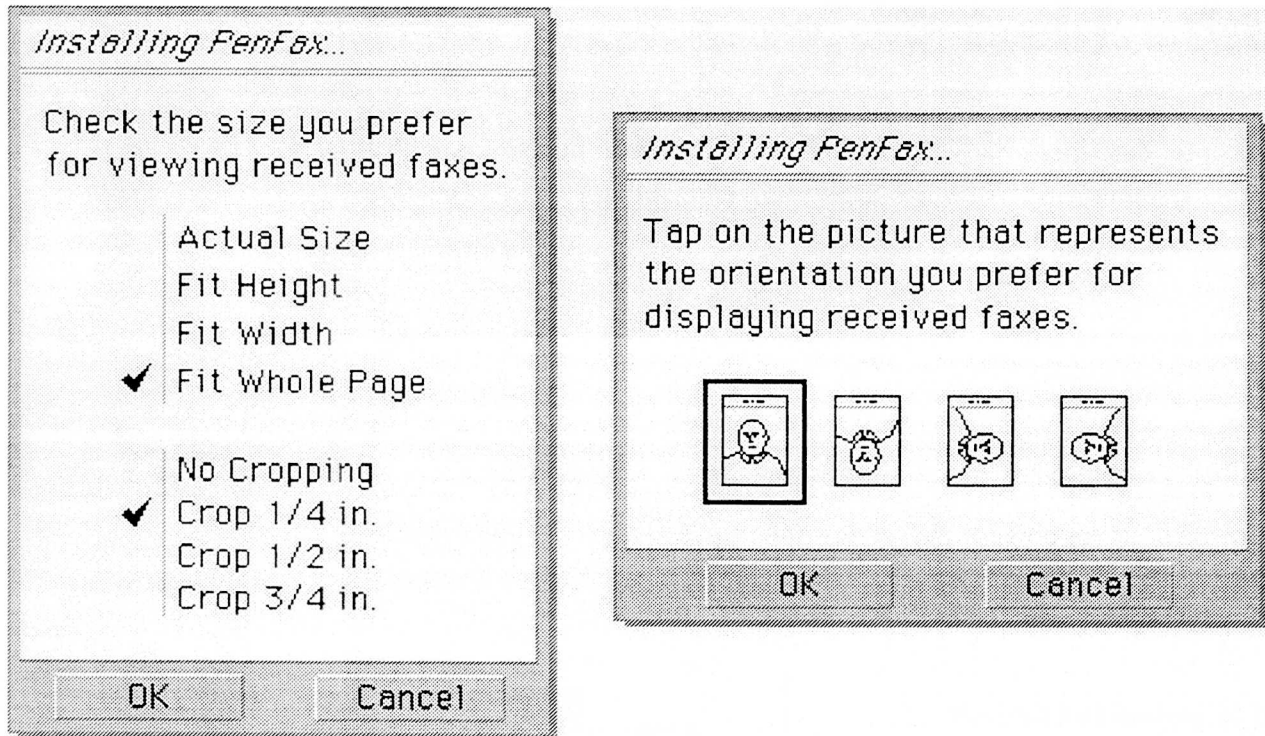
In box services such as fax and email don't fit into the stationery model. In the case of such services, new documents appear automatically in the In box, rather than being explicitly created by the user.

Once the document has been received, the user can set options for the document in the usual way, using the document's option sheets. But you need to provide a way for the user to set options for documents that will be received in the future.

The place to present such options is on the option sheet for the service in the Installed Services page of the Settings notebook, as described in "Global Application Options" in Chapter 8.

If you judge that the options are important enough, you may want to also present them to the user at installation time. By doing so, you give the user the opportunity make important configuration decisions as part of the installation process, while also permitting modification of the option settings at any time after installation (from the Settings notebook).

Take, for example, a fax service that allows the user to set the size and orientation for viewing the received fax document. You could present these options to the user on modal dialog sheets that pop up after the user has initiated the installation process:



Installation Options Dialog

Flexibility versus Simplicity

When considering whether to present configuration options at installation time, bear in mind that adding such options has the undesirable effect of making the installation process more complex.

In the printer driver example on the preceding page, presenting the list of printers during installation is strongly recommended, since the step of specifying the desired printer must be completed before anything can be printed.

In the fax example above, on the other hand, the presentation of options during installation is not as important, because the user can use the application without changing the default settings.

Chapter 5 / Managing Embedded Objects

This chapter describes what you should do to make use of the PenPoint™ operating system's support for live application embedding.

By supporting embedding, you make your application more valuable by allowing your users to embed not only documents, but also reference buttons, signature pads, and any other kind of component that GO or other PenPoint developers provide in the future.

▀ Basic Support for Embedded Objects

To provide basic support for the display of embedded objects, your application doesn't need to distinguish between the different types of objects (documents, reference buttons, and so on).

▀ Storing Embedded Objects

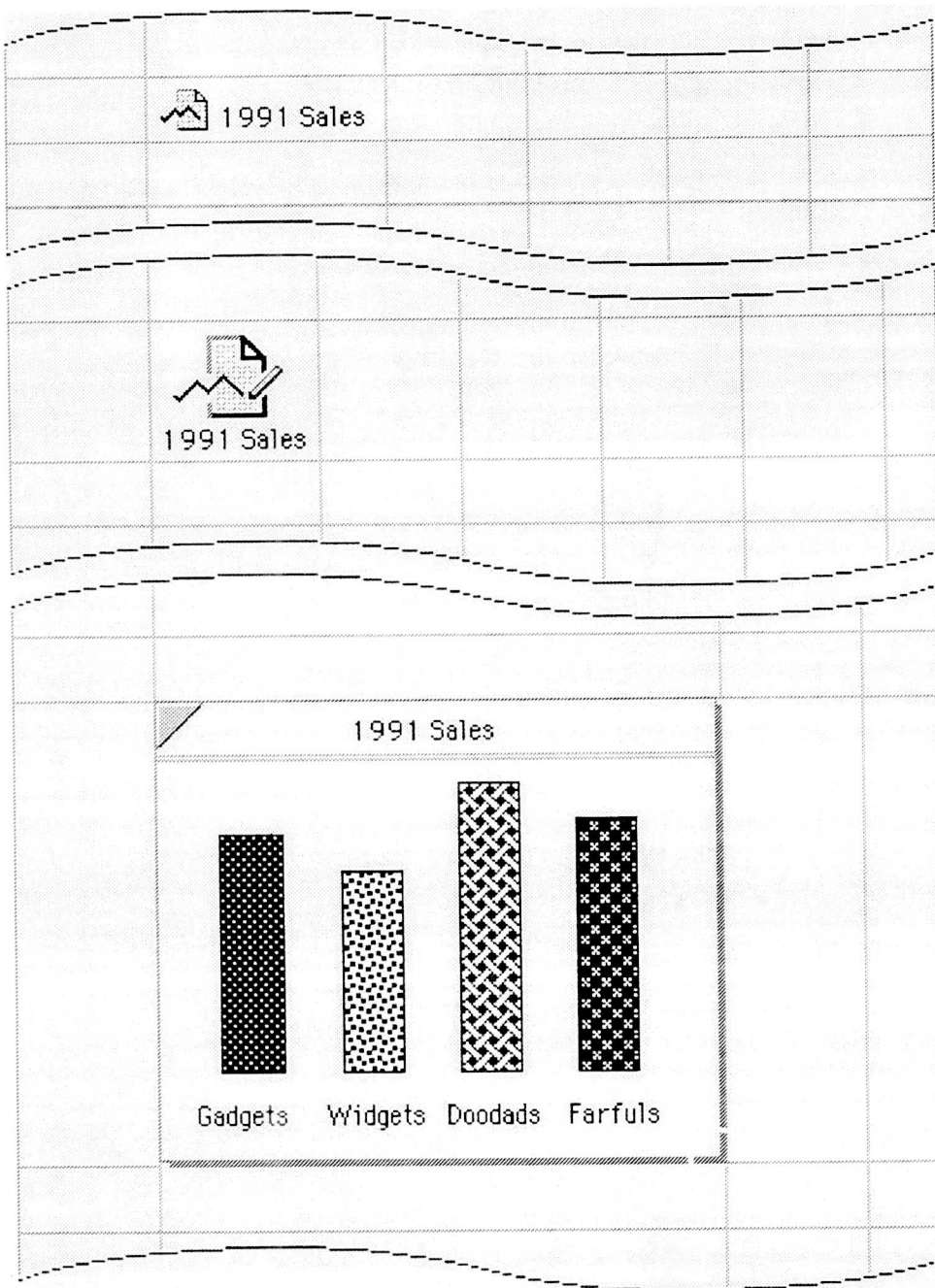
It is important to think about embedded objects from the *beginning* of your design process, so that you design your application's data structures to support embedded objects. A spreadsheet must be able to store an object as well as a formula in a cell; a word processor must be able to store an object in a character position; a drawing program must be able to manipulate objects that are embeddees as well as native objects, and so on.

▀ Displaying Embedded Objects

By responding to the embedding protocol, your application can track embedded objects as they come and go and change size. Your application should "make room" for embedded objects.

Exactly what this means will depend on the application. Let's take as an example a hypothetical spreadsheet that takes a simple approach: The embedded object always occupies a single cell, and the row and column size of the cell shrink and expand to match the size of the object.

Here are three views of the spreadsheet with an embedded chart:



Embedded Chart in a Spreadsheet

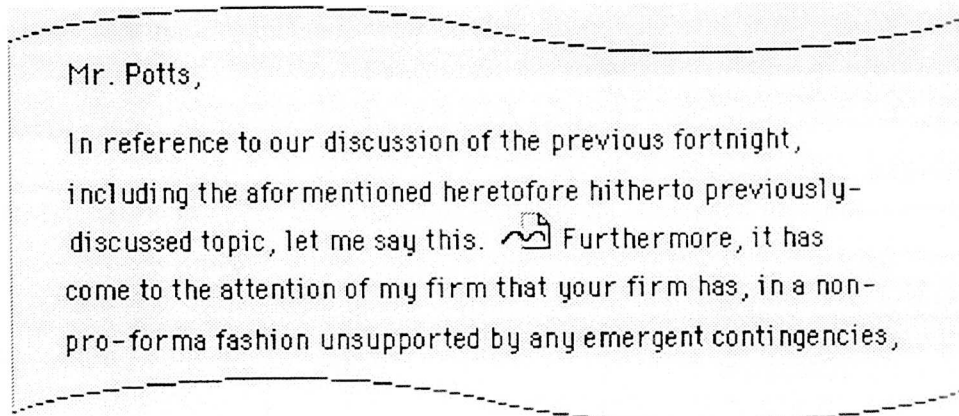
The top view shows the default icon style. In the middle view, the user has chosen a different icon style from the icon's option sheet. In the bottom view, the user has opened the chart.

▼ Intelligent Layout of Embedded Objects

The next level of support for embedded objects requires the application to distinguish between different types of embedded objects in order to optimize layout strategies.

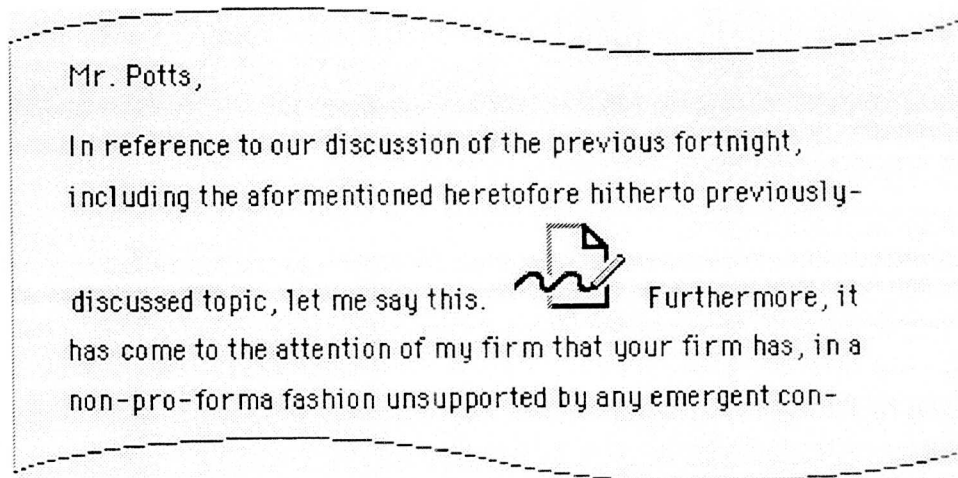
▼ Special Handling of Open Documents

The next three figures show an embedded notetaker document used as an annotation in a word processor. Here is the note in its iconic form:



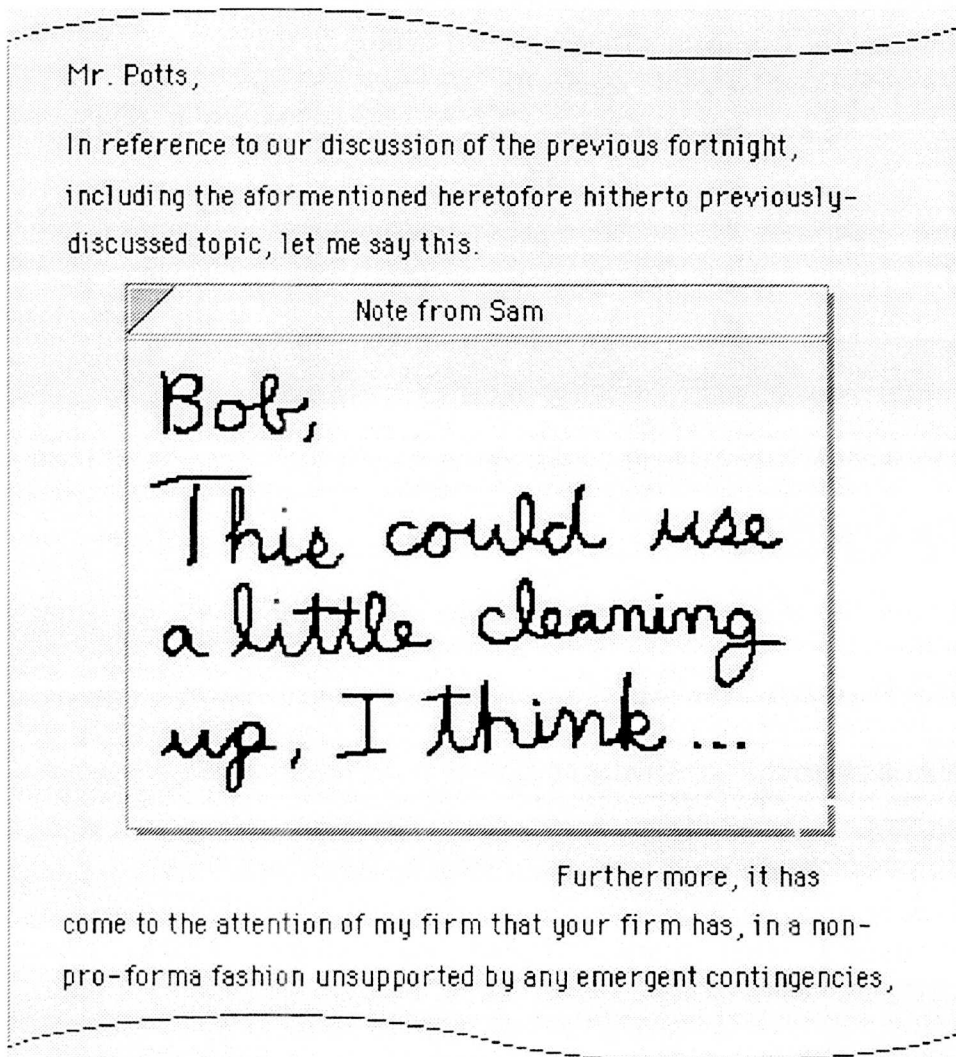
Embedded Icon in Text

The word processor treats the icon as a single character in the text stream. No special treatment is required when the user changes the icon style:



Large Embedded Icon in Text

The word processor does treat the open document differently from the icon, however. The application centers the open document on its own line:



Note that this kind of special handling requires that the application be aware of the type of the embedded document. In the above example, the word processor must apply the special handling only to documents, and only to documents that are open.

⚡ When Opening In-Line Is Not Appropriate

The PenPoint Application Framework supports two forms of open embedded documents: **in-line** and **pop-up**.

In the in-line style, PenPoint clips the embedded document to the containing document, and the embedded document appears to be in the same plane as the other visual objects in the containing document. The examples given previously in this chapter (of the chart embedded in the spreadsheet and the note embedded

in the word processor) have been concerned with the treatment of the embedded document when it is opened in-line.

However, the in-line style may not make sense in the context of some types of applications. For example, it may be very useful to allow the user to drop an icon into a field on a form. But opening in-line would require the information on the form to reformat around the open document, which is stretching the familiar concept of a form quite a bit.

In the imaginary form shown below, the user has embedded two icons in the Comments field:

Loan Application		
Name <i>Fast Eddie Soloko</i>		
Address <i>12 Musk Ox Crossing</i>		
City <i>Sicily</i>	State <i>Alaska</i>	Zip <i>40432</i>
Employer <i>Lagomorph Apparel, Ltd.</i>		
Comments  Credit Report  Sharon's Notes		

Form with Embedded
Pop-Up Note (Closed)

When the icon opens, it pops up over the form, without affecting the form's layout:

Loan Application		
Name <i>Fast Eddie Soloko</i>		
Address <i>12 Musk Ox Crossing</i>		
City <i>Sicily</i>	State <i>Alaska</i>	Zip <i>40432</i>
Employer <i>Lagomorph Apparel, Ltd.</i>		
Comments		

Sharon's Notes

Haven't been able to reach Mr. Soloko in 2 weeks...

Form with Embedded
Pop-Up Note (Open)

Normally, the user can choose either the in-line or the pop-up style for any embedded icon from the icon's option sheet. If you turn off the in-line style for documents embedded within your application, PenPoint will automatically disable the in-line option on the icon option sheet.

▼ **Marking Locations within a Document**

Another important aspect of PenPoint's Embedded Document Architecture has to do with marking arbitrary locations within a document. The PenPoint operating system includes a set of messages that together comprise a **mark** protocol.

✦ **Reference Buttons**

By responding to the mark protocol, you allow the user to select an object within your application and make a reference button linked to it. This allows your application to participate in one of the unique aspects of the notebook user interface: Namely, that the user can create customized access paths to data anywhere in the system.

Here are just a few of the many ways users can use reference buttons:

- ◆ Place buttons on the Bookshelf to mark frequently referenced information.
- ◆ Place buttons in one document that reference related documents.
- ◆ Create a customized index by arranging reference buttons in a MiniText or MiniNote document.
- ◆ Place reference buttons in the cork margin of a long document to scroll quickly to commonly accessed locations.

✦✦ **Scrolling the Marked Location into View**

When the user taps the reference button, the PenPoint operating system will turn to your document; it is up to you to scroll the marked location into view.

For information that is laid out sequentially, such as text, it is usually best to scroll the marked location to the top of the viewing region. With other types of documents, such as drawings or maps, it may be better to center the marked location.

Providing the Button Label

You also need to provide the label for any reference button linked to a mark in your application. In some contexts it is clear what to use as the label; MiniText, for example, simply provides the first 32 characters of the selection.

However, it may not be so clear what to use as the label. Take the example of a reference button linked to a cell in a spreadsheet. You could simply always give the cell's identifier (such as "C14"). On the other end of the spectrum, you could add a control to the reference button's option sheet allowing the user to choose the identifier of the cell, the cell's text label, the formula associated with the cell, or the cell's current value.

Spell, Proof, and Find

The **Spell**, **Proof**, and **Find** commands also rely on the mark protocol to navigate through trees of embedded documents. By responding to the mark protocol, you also go most of the way to supporting these functions.

Part 3

User Interface Building Blocks

DESIGNING A PENPOINT APPLICATION

PART 3 / USER INTERFACE BUILDING BLOCKS

Chapter 6 / Controls	175	Layout Issues	206
Buttons	175	Standard Layout	206
Button Labels	175	Non-Standard Layouts	206
Button Styles	175	Pop-Up versus In-Line Choices	207
Command Buttons	178	Deactivating versus Hiding Controls	209
Toggle Switches	180	Chapter 9 / Status, Warning, and Error Feedback	210
Lists	181	Wording Guidelines	210
Text versus Pictures	181	Wording of Messages	210
Checklists versus Boxed Lists	182	Wording of Button Labels	210
Pop-Up List Variations	183	Progress and Completion Messages	211
Multiple Lists	184	Confirmation Notes	212
Zero or One Lists	185	Error Notes	212
Checklists with Fields	186	Timing-Triggered Notes	212
Scrolling Lists	186	Audible Feedback for Warning and Errors	213
Text Fields	188	Message Lines	214
Font	188	Chapter 10 / Help	215
Fill-In versus Overwrite Fields	188	Quick Help	215
Edit Pads	189	Help Notebook	215
Invoking Edit Pads	189	Chapter 11 / Designing Application Icons	216
Keyboard Focus	189	GO's Conventions for Icons	216
Customizing Edit Pads	189	Icons for Documents	216
Input Behavior	190	Icons for Accessories	217
Chapter 7 / Menus	192	Icon Dimensions and Hot Point	217
Standard Menus	192	Icon Design Guidelines	219
Document Menu	193	Start with the Small Size	219
Edit Menu	194	Keep It Simple	219
View Menu	196	Relationship of the Small and Large Bitmaps	220
Menu Layout	196	Don't Use the "3-D" Style	221
Multi-Column Menus	196	Designing the Icon Mask	221
Separating Groups of Controls	197	Using Icons to Show Application State	222
Two-State Switches in Menus	198	Chapter 12 / Putting the Building Blocks Together	223
Fields in Menus	199	Basic Guidelines	224
Hierarchical Menus	199	Use of Gestures	224
Chapter 8 / Option Sheets and Dialog Sheets	200	Layering	224
Option Sheets	200	User Configurability	224
Adding to the Default Document-Wide	200	Where to Put It: Menu, Option Sheet, or Palette Line?	225
Option Sheets	200	Strengths and Weaknesses of Each Mechanism	225
Adding a New Document-Wide Option Sheet	202	How to Decide Which Mechanism to Use	226
Dynamic Behavior of Modeless Option Sheets	203	When to Depart from the Standard Building Blocks	227
Global Application Options	204		
Dialog Sheets	205		
Single and Multiple Dialog Sheets	205		
Modeless and Modal Dialog Sheets	205		

Chapter 6 / Controls

This chapter gives guidelines for using buttons, lists, fields, and edit pads.

■ Buttons

This section covers conventions related to buttons, including button labels, command buttons on option and dialog sheets, and when to use the different visual styles of buttons.

⚡ Button Labels

Follow these conventions, which have become *de facto* standards for graphical user interfaces:

- ◆ **Use verbs.** When possible, use verbs for both the label of the menu and the commands contained in the menu.
- ◆ **Capitalize.** Capitalize the first letter of each word of both the label and the contents of the menu.
- ◆ **Ellipsis convention.** If the command requires additional information before it can be completed, add an ellipsis (three dots) to the button's label. Examples include a **Delete...** command that pops up a confirmation note, a **Zoom...** command that displays an adjustable rectangle to define the zoom region, and the PenPoint **Move...** and **Copy...** commands, which surround the currently selected object with a marquee indicating that the user can drag it.
- ◆ **Deactivating button labels.** If a command is unavailable for any reason, set the button to the inactive state. The toolkit will render it in light grey.

⚡ Button Styles

This section gives guidelines for when to use the various styles of buttons.

☞ Raised Buttons on Grey Background

The raised button style is appropriate when you intend to produce the effect of a mechanical control panel, as in this keyboard accessory:

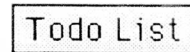


Raised Buttons on Grey Background

Do *not* use raised buttons on ordinary notebook documents, or on any surface that the user would otherwise perceive as flat and paper-like. The raised buttons produce a strong visual effect that works against the notebook and paper metaphor of the PenPoint™ operating system.

☞ Outlined Buttons on White Background

PenPoint uses this style for user-created reference buttons:



Outlined Button on White Background

Use outlined buttons when you have one or two buttons on a white, paper-like surface.

☞ Square Buttons with Long Labels

This form works better than a full outline for buttons with long labels, especially when they are arranged in a column rather than a row:

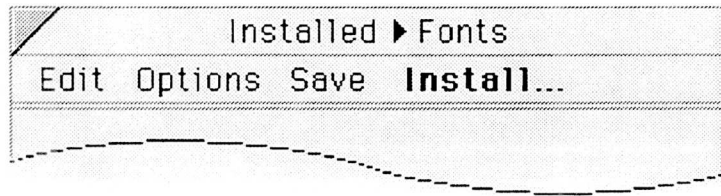
- ☐ Lower Case Letters
- ☐ Upper Case Letters
- ☐ Numerals & Symbols
- ☐ Individual Characters
- ☐ Exit

Square Buttons with Long Labels

✦ Bold Buttons in Menu Lines

If you have a command that is very important or frequently used, consider putting it right on the menu line, rather than within a menu. That way novices can't miss it, and it's more efficient to access for frequent users.

Put the button to the right of the menus on the menu line, and make its label bold to distinguish it from the ordinary menu labels:



Bold Button in Menu Line

Note that in the above example the **Install...** button has an ellipsis because it brings up a dialog sheet.

✦ Half-Outlined Buttons on White Background

The half-outlined button is designed to facilitate the use of buttons in the context of surrounding text. The open left end allows the button to be read as part of the flow of the text, while the curved right distinguishes the button from an underline, and gives the suggestion of a button. The example below shows help text with embedded reference buttons:

PenPoint features two forms of help. You can get Quick Help at any time by simply tapping on the Help icon and then tapping on any object visible on the screen to see a brief message about the object. Or, you can open the Help Notebook and browse through it. The Help Notebook contains information, organized by topic, for both PenPoint itself and for each installed application.

Half-Outlined Buttons in Text

✦ Customizing Buttons

You can modify the shape and border style of buttons to obtain visual effects other than those already described. Follow these general guidelines when customizing button styles:

- ♦ Don't depart from the standard button styles just to be different. Usually, minor visual variations do not add substantial value to an application—but they do present one more thing for the new user to become familiar with. By using the standard styles, your application will fit in better with the overall PenPoint environment.

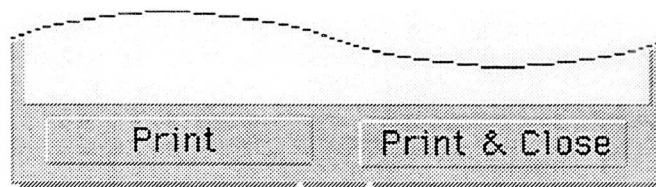
- ◆ If you do depart from the standard styles, pick a single style and use it consistently.
- ◆ Avoid the temptation to overuse heavy borders and thick shadows. Heavy borders and shadows tend to degrade readability because they draw attention away from the button's label and interact with the letters in the labels to set up competing visual shapes.

✦ **Command Buttons**

The light grey area at the bottom of dialog and option sheets is called the **command line**, and the buttons found there are called **command buttons**. This section describes conventions for the placement and labeling of command buttons.

✦ **Standard Modeless Buttons**

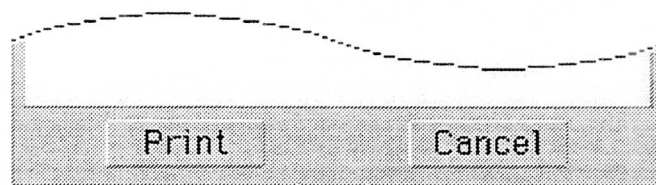
If the sheet is modeless and has a single command, use two buttons: one to execute the command and leave the sheet up, the other to execute the command and close the sheet. This convention takes advantage of the modeless nature of the sheet by allowing the user to use the sheet one time or repeatedly. The buttons should be ordered and labeled as in this example:



Standard Modeless
Command Buttons

✦ **Standard Modal Buttons**

For modal sheets having a single command, use two buttons: one to invoke the command and dismiss the sheet, and another to dismiss the sheet without taking any action. The latter button should always appear on the right, and be labeled **Cancel**. So, for example, if the **Print** sheet were modal, its buttons would be labeled as follows:



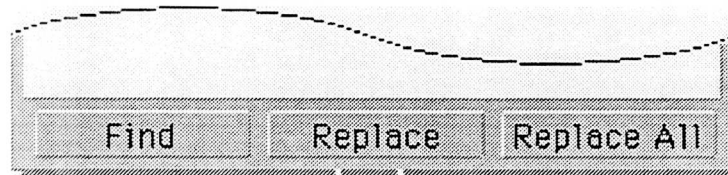
Standard Modal
Command Buttons

The word "cancel" is appropriate in two senses: The button cancels both the mode and the pending operation. The close/cancel convention also serves the

function of giving the user a consistent cue as to whether a given sheet is modal or modeless.

Non-Standard Buttons

Sometimes the standard conventions described in the previous sections aren't sufficient. The basic command for a dialog sheet may have variations, as in this example from the **Find** sheet:



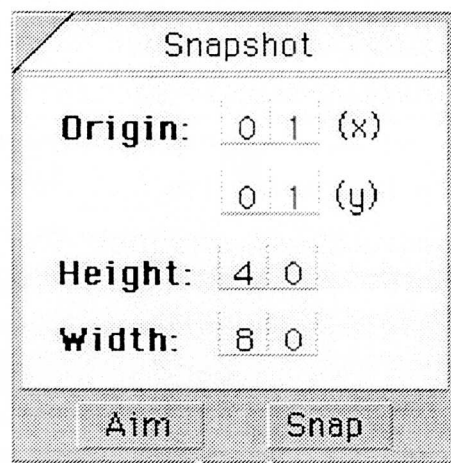
Non-Standard
Modeless Command
Buttons

In such cases, do the best you can to keep the command line simple. If possible, limit the buttons to a single row. If there are more buttons than will fit on a single row, consider grouping them into multiple sheets.

What Constitutes a Command Button?

Only buttons that actually affect something outside of the sheet should appear on the command line. Any other buttons should be placed within the body of the sheet.

Take the example of a screen snapshot program. There is a need for a button that the user can tap to go into “drag mode” to specify the area to be captured in the snapshot—let's call it the **Aim** button. It might seem at first glance that this button should go on the command line:

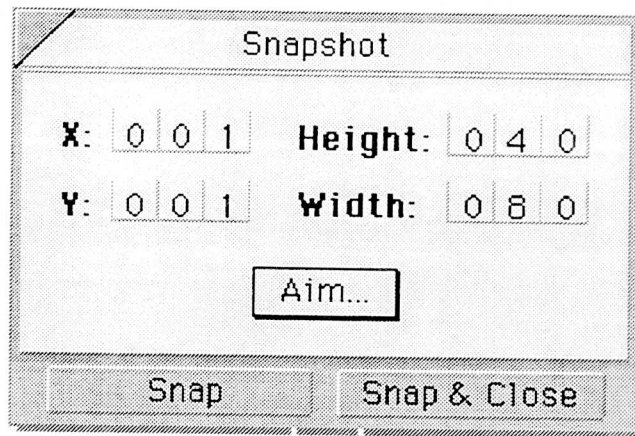


X Not Recommended
Non-Command Button
on Command Line

This design looks fine, but has the weakness of losing the distinction between buttons (like **Snap**) that actually *effect* the command and those (like **Aim**) that only *set up* the command. Even if the user never consciously thinks in terms of

this distinction, it is important to maintain it consistently, because it reflects a consistent model that underlies the interface.

The figure below shows a better solution:



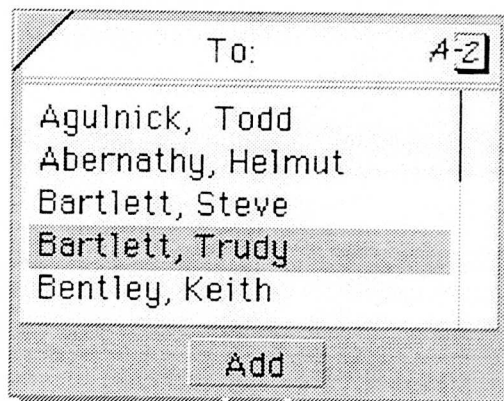
Proper Use of Command Line

The **Aim** button is in the body of the sheet, close to the parameters that it modifies. This also makes room in the command line for the standard two buttons.

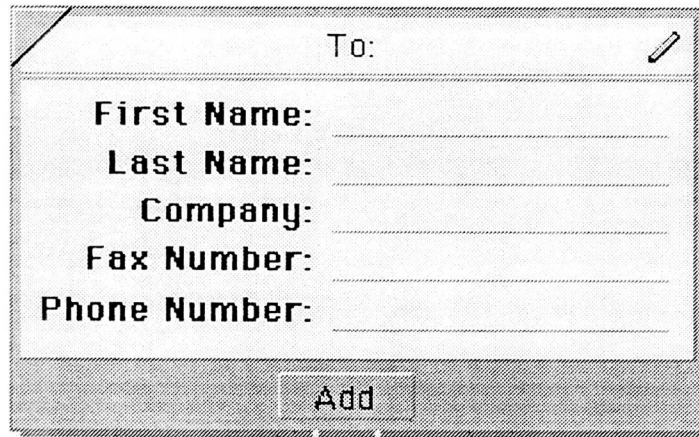
▼ **Toggle Switches**

Toggle switches are useful anytime you have two modes, states, or alternatives to present to the user.

The figures below show a sheet for addressing a fax document. The user can either pick from names drawn from an address book accessory or write the name of the addressee directly into fill-in fields. A toggle switch at the right of the title line lets the user choose between the list of names or the fill-in fields:



Toggle Shows Address Book Icon when Address Book Entries Are Shown



Tapping the Toggle
Switches to Fill-In
Fields

Note that in the example above, the toggle reflects the *current* state, *not* the state that tapping on the toggle will bring about.

▼ Lists

The PenPoint User Interface Toolkit provides several types of controls for presenting lists to the user. This section gives guidelines for using lists, including when to use pictures instead of text for the choices in the list, when to use boxed lists instead of checklists, and how to use pop-up lists to save space.

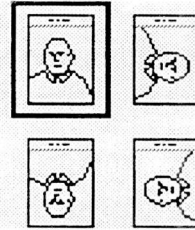
▼ Text versus Pictures

You can specify that the individual choices be represented either as text strings or as pictures. (NOTE: Small pictures used in GUIs are often referred to as “icons.” This document uses icon in the more restricted sense of the icon associated with each application that represents the type of the document or accessory.)

While lots of pictures may be perceived by some as adding “sizzle” to an interface, remember that a given picture is just as likely to impede communication as to promote it. Exercise restraint when using pictures; use them *only* when you are satisfied that they communicate more effectively than text.

In the example below, the pictures are clear and unambiguous, and help make the control understandable at a glance:

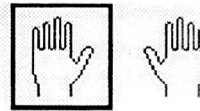
Orientation:



*Appropriate Use of
Pictures*

In the example below, on the other hand, the pictures are nearly useless:

Hand Preference:



X Not Recommended
*Inappropriate Use of
Pictures (Which Hand is
Which?)*

It is far better in this case to simply spell out the choices:

Hand Preference: ✓

Right Hand
Left Hand

Appropriate Use of Text

⚡ **Checklists versus Boxed Lists**

The PenPoint operating system uses vertical checklists as the default form for in-line lists in dialog and option sheets because they are simple, paper-like, and easy to scan:

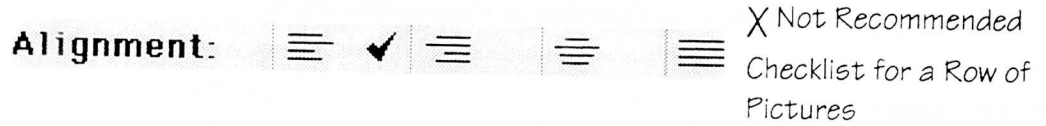
Alignment: ✓

Left
Right
Center
Full

Basic Vertical Checklist

Checklists don't work as well if you are laying out the list horizontally, or in a matrix, in order to use the space on a sheet more efficiently. In non-vertical layouts, it's not as clear which choice the check is associated with, and the vertical grey line doesn't tie the choices together as it does when they are in a column. Checklists are also not ideal for lists of pictures because the check doesn't stand out as well as it does when the choices are text.

The figure below, showing a list that is non-vertical and non-textual, illustrates both of these problems:

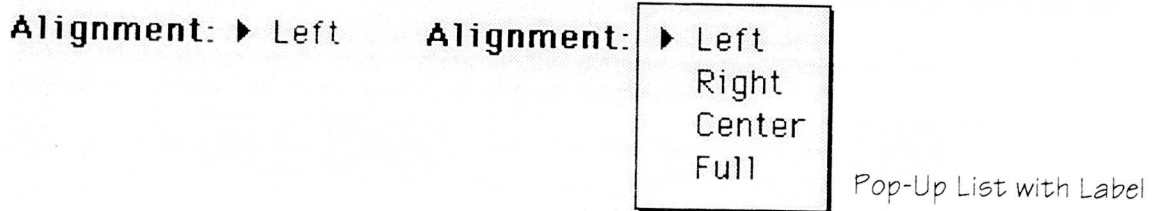


Using the boxed style makes the same list both clearer and more compact:

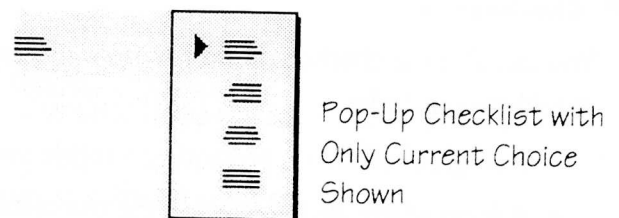


Pop-Up List Variations

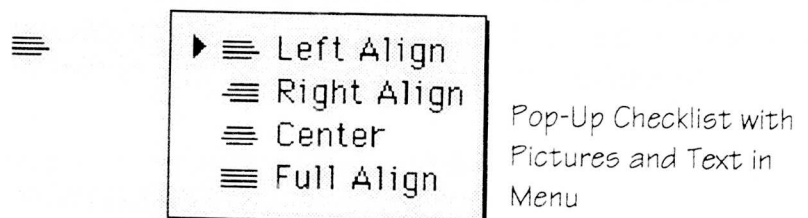
In dialog and option sheets, pop-up lists typically have a text label on the left:



For contexts in which space is at a premium, such as palette lines, you can make the list even more compact by using pictures for the choices and suppressing the display of the label and arrow:



To make the list both compact and easy to interpret, you can specify that the pictures be augmented by text on the menu:

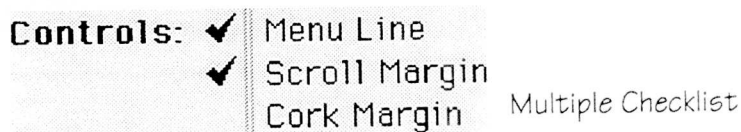


Multiple Lists

For multiple choice lists, you can use checklists, boxed lists or checkboxes. Each of these styles has its place, as described in the sections below.

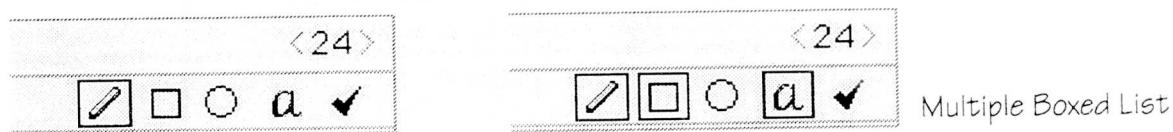
Checklists

For multiple lists in option sheets, dialog sheets, and menus, use the checklist with a double vertical line:



Boxed Lists

Use the boxed form for multiple lists in horizontal contexts:



Boxed checklists are the ideal control to use in constructing palettes and control margins, as described in Chapter 12, "Putting The Building Blocks Together," and Chapter 16, "Presenting Input Modes."

Checkboxes

You can also use checkboxes for multiple choice lists. Situations that call for checkboxes include:

- ◆ A single checkbox representing a toggle switch.
- ◆ A form where checkboxes are the common convention.
- ◆ A column of checkboxes in a table.

The PenPoint operating system uses checkboxes in the notebook table of contents to provide an easy way for the user to toggle tabs on and off for individual sections and documents:

	Clients	3	☒	
	East	4	☐	
	West	5	☒	
	North	6	☒	

Checkboxes for Non-Exclusive Choices

Zero or One Lists

Most lists fall neatly into the single choice category, in which exactly one choice is on at any given time, or the multiple choice category, in which the user can turn each of the choices on or off independently.

Occasionally, the semantics of a list are such that the user can choose either one or none of the choices. In this situation, you can either add an explicit choice to represent “all choices off” or you can present a variation of a single choice list in which tapping on a choice that is on turns it off (leaving no choices on).

The notebook table of contents presents such a situation. For each entry, the user can choose to show the entry’s icon (which is application-specific), a simple square button (which is more compact), or neither (which is the least cluttered).

The **Show** control from the table of contents **Layout** option sheet lists only two choices: **Icons** and **Buttons**. In the figure below, the user could either tap **Buttons** to move the check to that choice, or tap **Icons** to remove the check altogether:

Show: ☒ Icons
☐ Buttons *Zero or One List*

The same semantics could be expressed by using an ordinary checklist and adding a third choice to explicitly represent the state in which neither the icon nor the button is shown:

Show: ☒ Icons
☐ Buttons
☐ Name Only *Checklist with an Explicit Choice for “None of the Others”*

The “zero or one” style of list has the advantage of being more compact, and the disadvantage of not explicitly presenting the “all choices off” alternative to the user.

Checklists with Fields

Checklists are not constrained to a single type of control.

The figure below shows a common idiom, in which one of the choices in a checklist contains one or more overwrite or fill-in fields:

Pages: ☒ All
From to

Checklist with Overwrite Fields

Scrolling Lists

With scrolling lists, one of the essential design questions to ask is, how will the information in the list be used? There are several usage patterns that suggest different variations, as described in the following sections.

Scrolling Checklists

When the purpose is to allow the user to indicate that exactly one of the items is “current,” or distinguished for some purpose, use a scrolling checklist:

Font: ☒ Courier
☐ Hand Print
☐ New York
☐ Sans Serif
☐ Terra Cotta

Scrolling Checklist

Scrolling Pop-Up Checklists

Use a scrolling pop-up checklist when you need to present choices in a more compact format:

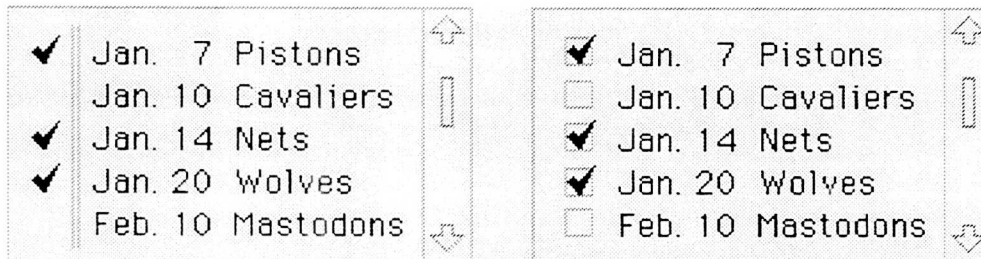
State: ▶ Delaware **State:** ▶ Delaware

California
Colorado
Connecticut
Florida
Georgia
Hawaii

Scrolling Pop-Up Checklist

Scrolling Multiple Checklists

When the purpose is to allow the user to choose more than one of the items in the list, the scrolling list behaves like a multiple choice control:

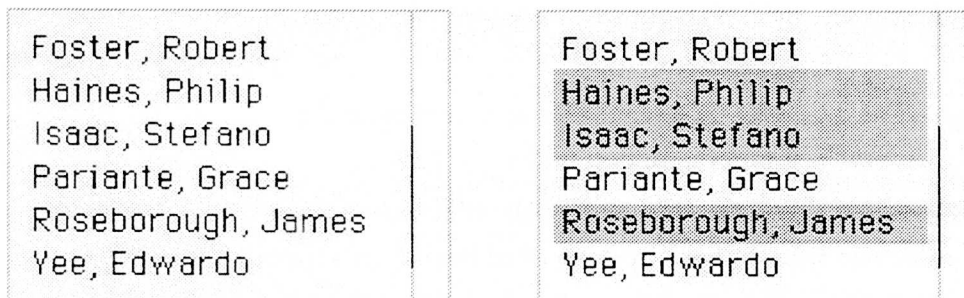


Scrolling Multiple Choice Lists

The list on the left uses the PenPoint double-line convention. The list on the right uses checkboxes. Use the double-line form if the list is in a standard dialog sheet or option sheet. Use checkboxes if you are using checkboxes for similar purposes throughout your application.

Scrolling Lists of Selectable Items

Often a list contains items that the user can edit; the basic operations are Create, Delete, Move, and Copy. While such editing operations in PenPoint can be invoked with gestures, you should support selection of the items, both to allow editing from a menu and also to support editing on more than one item at a time. The figure below shows a list of names for an address book. None of the names are selected in the left-hand list; three names are selected on the right:



Scrolling List of Editable Items

In such editable lists, follow the guidelines for selection presented in Chapter 20. If only one item in the list can be selected, tap is the only selection gesture to support. If multiple items can be selected, you should implement the full set of selection gestures, including the brackets and press-hold to select contiguous items, and plus to select discontinuous items.

Also, the selection feedback should appear *after* the selection gesture is recognized, *not* on pen-down. If the feedback appears on pen-down, it interferes visually with the gesture that the user is trying to make.

✦ Gesture Accelerators for Scrolling Lists

In long, alphabetically sorted lists, it is a good idea to use the capital letters as accelerators for scrolling. Allow the user to write the letter anywhere on the list. When a letter is recognized, select the next item in the list beginning with that letter. If the item is not visible, scroll the list so that the item appears at the top of the viewing region. If the letter does not appear as the first letter of an item, scroll to the letter following it.

✦ Text Fields

This section gives guidelines for using fill-in and overwrite fields.

✦ Font

By default, fill-in fields use the font specified as the **Field Font** preference, and the size specified as the **Font Size** preference.

Your application can override these preferences, either for the purpose of fixing the font and size to ensure that all the information on a sheet can be viewed at once, or because your application provides its own user interface for controlling fonts. See “Allowing the User to Specify the Font” in Chapter 18 for related guidelines.

✦ Fill-In versus Overwrite Fields

The PenPoint User Interface Toolkit provides two styles of text fields: underlined fill-in fields and overwrite fields that are segmented into discrete character boxes.

Each style has its advantage. The overwrite fields are easier to use, because the user can edit them directly without first tapping to pop up an edit pad.

The disadvantage of the overwrite fields is that they are usually much larger than fill-in fields. The sizes of these two types of field are controlled by different user preferences: Fill-in fields are shown in the size of the system font, while overwrite fields take their size from the **Box Size** preference on the Handwriting Preference sheet. By default, the overwrite fields are larger, to make it easy to write accurately in them.

By default, PenPoint sheets use overwrite fields for numeric fields and fill-in fields for text fields, which are usually longer.

Given the tradeoff between editing efficiency and compactness, it is up to you to decide which style to use. Unless there's a compelling reason to mix them, use only one style on a single sheet.

▼ Edit Pads

This section covers using and customizing the standard PenPoint edit pads.

⚡ Invoking Edit Pads

The circle is the core PenPoint gesture for the basic **Edit** operation. Drawing a circle should always bring up the edit pad for the object under the circle.

Tapping should also display the edit pad, unless there is a more important operation that should take precedence. For example, tapping on a fill-in field in a dialog or option sheet displays the edit pad for the field, because editing is the primary operation on the field. But tapping on a tab or reference button turns to the associated document, and tapping on a title in the table of contents selects the document. In these cases, editing is not the primary operation.

The object being edited should take the selection, so that the user can see at a glance what the target of the pad is.

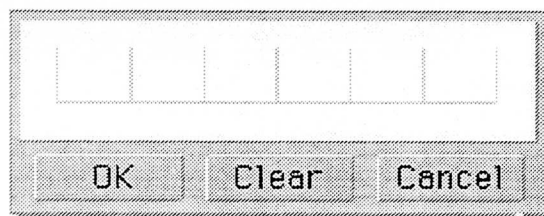
The text being edited should appear in the pad's overwrite boxes, with a couple of blank boxes at the end to allow the user to add one or two characters without resizing the pad.

⚡ Keyboard Focus

The first box of the edit pad should take the input focus, so the user can use the pad by typing. Pressing the **Enter** key should be taken as equivalent to tapping **OK**.

⚡ Customizing Edit Pads

The PenPoint User Interface Toolkit provides standard pop-up writing and edit pads:



Pop-Up Edit Pad

To make editing in your application as convenient and efficient as possible, you may want to design special edit pads that are specialized to the types of objects in your application.

In designing these pads, remember the principle of the dual command path, and allow the user to enter information either through the handwriting engine (gestures or characters) or by tapping (buttons, menus, and scrolling lists).

Input Behavior

The standard pop-up pads are modal; while they are displayed, input to other parts of the screen is blocked, and they automatically take the focus of keyboard input.

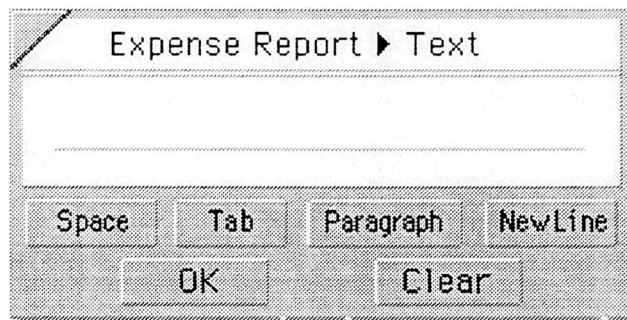
If you build an edit pad with more functionality, consider making the pad modeless, like the PenPoint option and dialog sheets, so that the user can leave the pad up and use it to edit several objects.

The price of making the pad modeless is that it has to handle the selection correctly. In particular, make sure that the command buttons are dimmed if there is no selection of the proper type.

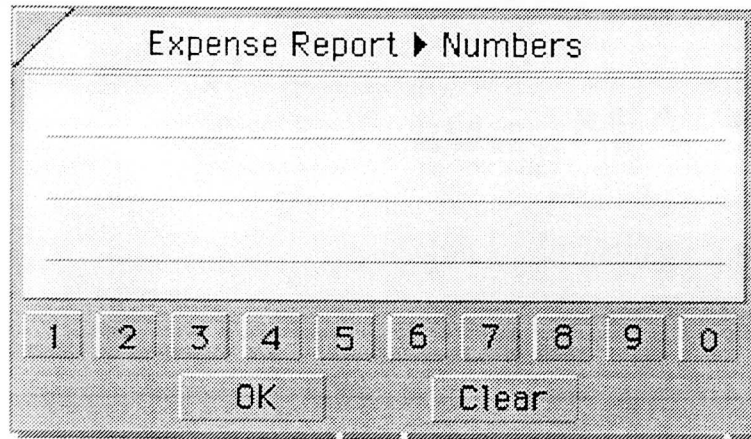
Example

The figures below show an example of a modeless pad that supplements the writing area with buttons for inserting characters directly into the document at the insertion point.

The user can choose from three layouts of buttons, optimized for editing text, numbers, or dates:



Edit Pad with Text-Related Buttons



Expense Report ▶ Numbers

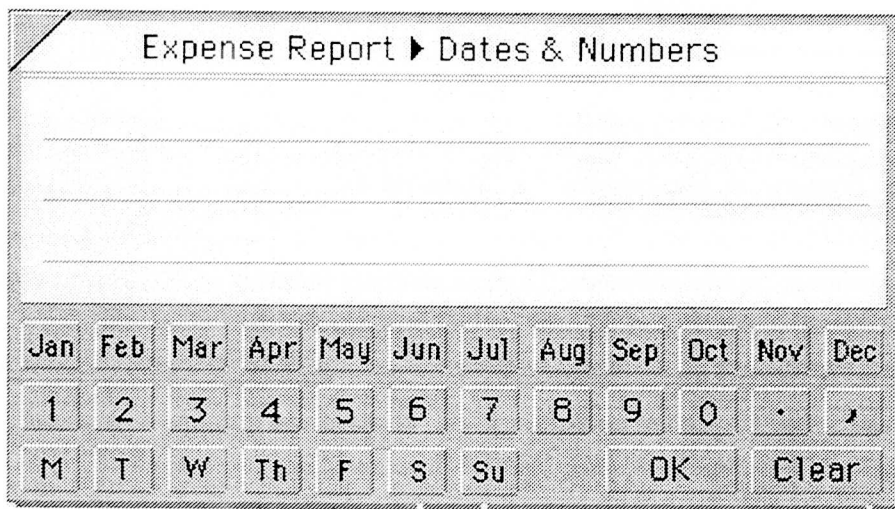
Three horizontal lines for text input.

1 2 3 4 5 6 7 8 9 0

OK Clear

This is a dialog box titled "Expense Report ▶ Numbers". It features three horizontal lines for text input. Below the input area is a numeric keypad with buttons for digits 1 through 0. At the bottom of the keypad are two buttons labeled "OK" and "Clear".

Edit Pad with Buttons
for Numbers



Expense Report ▶ Dates & Numbers

Three horizontal lines for text input.

Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec

1 2 3 4 5 6 7 8 9 0 . ,

M T W Th F S Su OK Clear

This is a dialog box titled "Expense Report ▶ Dates & Numbers". It features three horizontal lines for text input. Below the input area is a date and time keypad. The first row contains buttons for the months of the year (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec). The second row contains buttons for digits 1 through 0, a decimal point (.), and a comma (,). The third row contains buttons for the days of the week (M, T, W, Th, F, S, Su) and two buttons labeled "OK" and "Clear".

Edit Pad with Buttons
for Dates and Numbers

Chapter 7 / Menus

This chapter gives guidelines for using menus, including:

- ◆ Customizing the standard menus.
- ◆ Conventions for laying out the controls in menus.
- ◆ Two-state switches in menus.
- ◆ Text fields in menus.
- ◆ Hierarchical menus.

▼ Standard Menus

The PenPoint Application Framework implements three standard menus: **Document**, **Edit**, and **Options**.

The entries in the **Options** menu are well-defined: There is an entry for each option sheet that is applicable when the menu is displayed. The **Options** menu never has checklists and is never hierarchical. For guidelines on option sheets, see Chapter 8.

The other two standard menus represent basic categories of functions that are widely applicable across applications:

- ◆ The **Document** menu is for commands related to the entire document, such as **Checkpoint**, **Revert**, **Print**, and **Send**.
- ◆ The **Edit** menu is for commands related to editing the objects in the document, such as **Move**, **Copy**, and **Delete**.

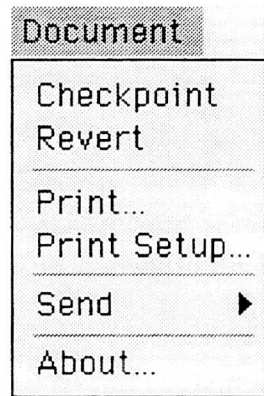
Follow these guidelines for using the **Document** and **Edit** menus:

- ◆ If the functionality covered by these menus is relevant to your application, use the standard menu labels unless there is a compelling reason to do otherwise.
- ◆ If you do use the standard menus, they should always be the first and second menus on the menu line.
- ◆ If any of the default commands on the standard menus are not appropriate for your application, remove them entirely (rather than deactivating them).

The sections that follow describe the standard menus.

Document Menu

The default Document menu contains six commands:



Default Document Menu

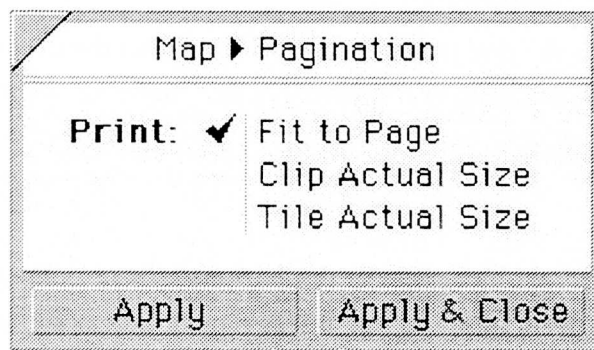
Checkpoint and Revert

The PenPoint Application Framework implements the **Checkpoint** and **Revert** commands for all documents that follow the normal PenPoint model and keep a second copy of their data in the file system. In fact, the PenPoint Application Framework automatically checkpoints such documents whenever the user turns away from the document's page.

You may choose to keep only one copy of your application's data. In that case, you need to either implement the **Checkpoint** and **Revert** commands yourself or remove them from the menu.

Print Setup and Print Preview

You can modify the contents of the three default **Print Setup** sheets, or you can add a new **Print Setup** sheet specific to your application, such as this one, which allows the user to specify how to print a drawing:



Application-Specific
Print Setup Sheet

If you have a **Print Preview** command, put it immediately after the **Print** command.

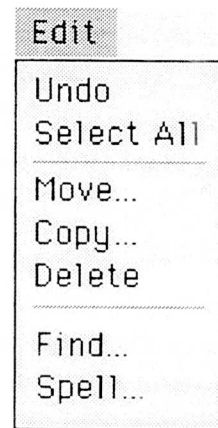
NOTE: In PenPoint 1.0, you cannot modify or replace the **Print** sheet. Future releases of PenPoint will support the customization of the **Print** sheet, as well as the **Find** and **Spell** sheets, which in PenPoint 1.0 can be replaced but not modified.

✦ **The About Application Sheet**

By all means customize the **About Application** sheet. This is the place to put your company logo and fancy artwork identifying your application.

✦ **Edit Menu**

The default **Edit** menu contains seven commands:



Default Edit Menu

✦ **Undo**

At a minimum, all applications should support a single-level undo. Modify the label of the command to indicate which operation would be undone (for example, **Undo Delete** or **Undo Move**).

Better than simple undo is the undo/redo model, which allows the user to redo the operation that was just undone. It is essential in this model to change the label of the command in order to make it clear to the user what the effect will be. For example, if the user taps **Undo Delete**, the next time the menu comes up, the command should read **Redo Delete**.

The multiple undo model, in which the user can backtrack by undoing editing operations one by one, is more powerful but also more complex to implement, document, and use.

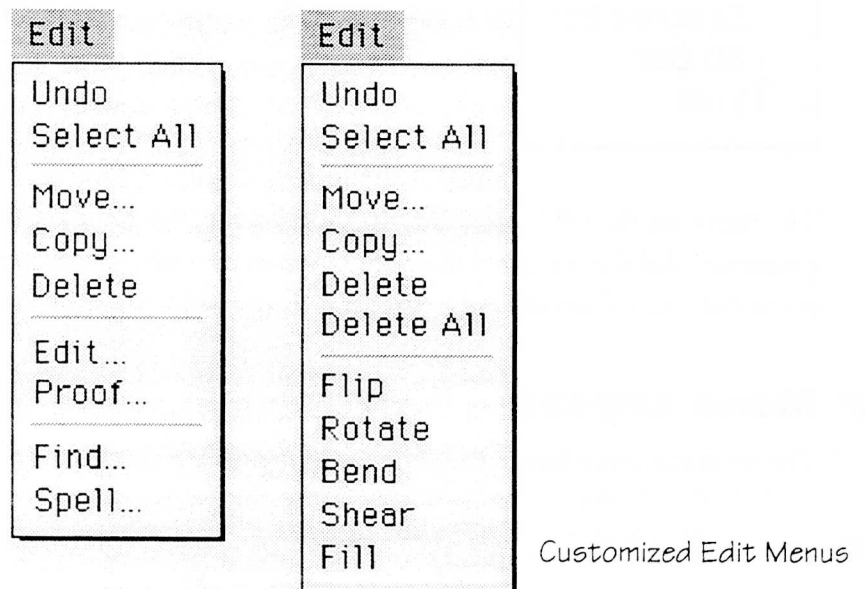
⚡ Create

Follow this guideline for commands that create new objects:

- ♦ If your application has a single command for the creation of new objects, put it in the **Edit** menu. An example is the **Connected Disks** page of the **Connections** notebook, which has a **Create Directory** command.
- ♦ If you have separate commands to create objects of different types, it usually makes more sense to present them on a separate menu. So, for example, the notebook table of contents has a **Create** menu for creating new documents and sections, and MiniText has an **Insert** menu for inserting new text and various other objects.

⚡ Examples

The figure below shows two typical application-specific **Edit** menus:



On the left is the menu for PenPoint's text component. The **Edit** and **Proof** commands have been added.

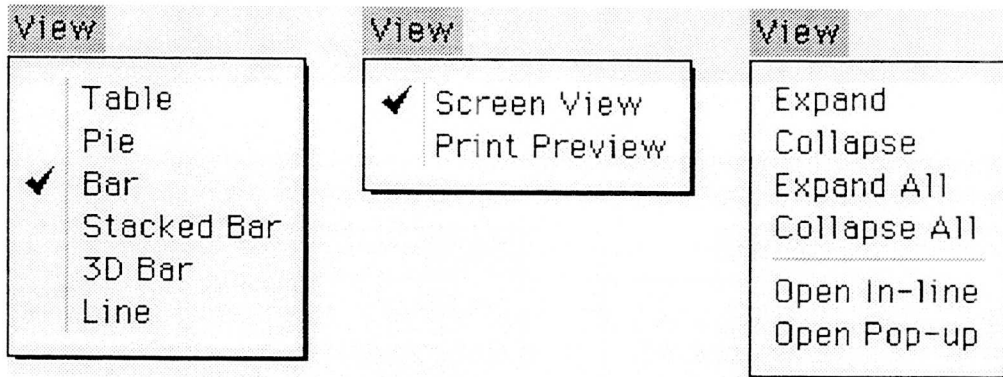
The menu on the right is for a drawing program. The **Find** and **Spell** commands, which don't apply to drawings, have been removed. A **Delete All** command has been added immediately below the standard **Delete** command. (This is the recommended wording and location, rather than calling it **Erase** or **Clear** and putting it on the **Document** menu.) Finally, several commands to act on figures have been added at the bottom of the menu.

View Menu

Whatever your application, you will probably have commands that allow the user to modify how the application's data is presented. You can present this information in a variety of ways: in menus, option sheets, or palette lines (Chapter 12 contrasts these interface building blocks).

If you do use a menu, consider calling it the **View** menu, and putting it immediately to the right of the **Options** menu.

Here are three examples of typical **View** menus:



Typical View Menus

The menu on the left is for a charting application, the middle menu is for a word processor, and the menu on the right is for an outliner, in which categories can be expanded and collapsed, and entries can be opened.

Menu Layout

The most common layout for controls in menus is a single column.

Multi-Column Menus

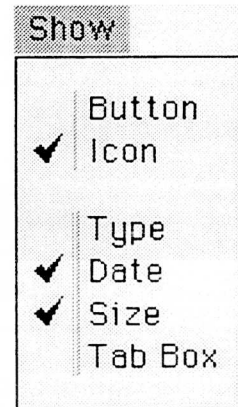
In some cases, the standard single column layout results in an extremely tall and skinny menu. An example might be a menu containing the numbers 1 through 20. In such cases, consider using a multi-column layout. The rule of thumb is to keep the menu a bit taller than wide—say a 5 to 3 ratio.

For readability, use a column-major sort order when the controls are sorted alphabetically or numerically. That is, the controls should read downward, in columns, rather than across, in rows.

⚡ Separating Groups of Controls

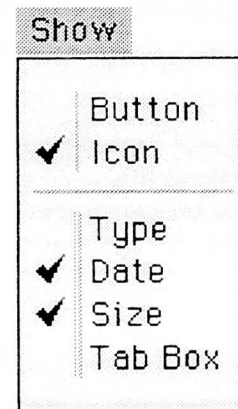
Separate groups of commands either with white space or with a light grey line.

Dividing lines are unnecessary between checklists in a menu, because the vertical single or double line unifies each list:



*Checklists in Menus
Separated with White
Space*

Adding a horizontal dividing line only makes the menu harder to scan:

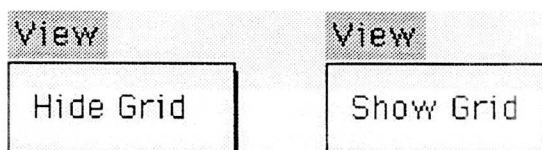


*X Not Recommended
Checklists in Menus
Separated with a
Dividing Line*

Two-State Switches in Menus

There are two recommended idioms for representing a two-state switch in a menu.

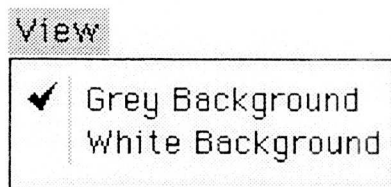
The first is to present the switch in the form of a single command whose label changes dynamically based on the state of the program. Use this method when the choices are obvious opposites. A typical example is a command to toggle a grid on and off. The command invites the user to hide the grid when the grid is on, and show the grid when the grid is off:



Single Command for
Two-State Switch

Another example is toggling the label of a single-level undo command between **Undo** and **Redo**.

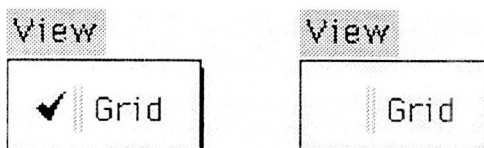
The second recommended method for presenting a two-state switch is to use a checklist with two items. Use this form when the choices do not constitute an obvious pair of opposites, as in this example:



Checklist for Two-State
Switch

Since the choices are not clearly opposites, it is clearer to show them both than to have a single command whose label changes from **Set Background to Grey** to **Set Background to White**.

Presenting a two-state switch as a multiple choice list with a single choice, as shown below, is *not* recommended:

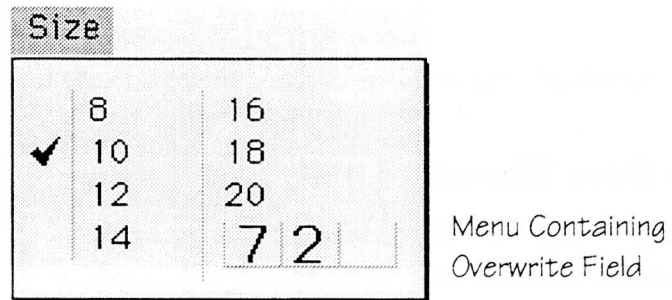


X Not Recommended
Multiple Checklist for
Two-State Switch

In this idiom, the state of the switch is indicated by the presence or absence of a checkmark. This method is not as clear as the two methods described above—a single command whose label toggles, or a checklist showing both choices.

Fields in Menus

A menu containing a checklist, one of whose elements is a text field, is a good way to present the most common choices to the user while also allowing less common values to be entered:



Note that writing into the field doesn't dismiss the menu.

Tapping to the left of the vertical line, by the field, chooses that item and dismisses the menu, as usual. But directing input to the field itself (entering a character, making an editing gesture, or tapping to set the keyboard input focus) chooses the item and leaves the menu up. This allows the user to complete the editing operation and verify that the contents of the field are correct before dismissing the menu.

Similarly, tapping on a fill-in field in a menu displays the field's edit pad, and dismissing the pad leaves the menu displayed.

Hierarchical Menus

As a general rule, keep the menu hierarchy to one level. Hierarchies deeper than a single level are cumbersome to use. You can always avoid a deep hierarchy by either making the hierarchy broader at the top level or putting commands on other types of controls, such as option sheets or palette lines.

Chapter 8 / Option Sheets and Dialog Sheets

This chapter gives guidelines for using dialog sheets and option sheets.

▼ Option Sheets

Use option sheets to allow the user to set properties of objects within the document, or for the document as a whole.

Chapter 5 of Book I describes the PenPoint™ operating system's option sheet model and how option sheets work. This section gives further guidelines for their usage.

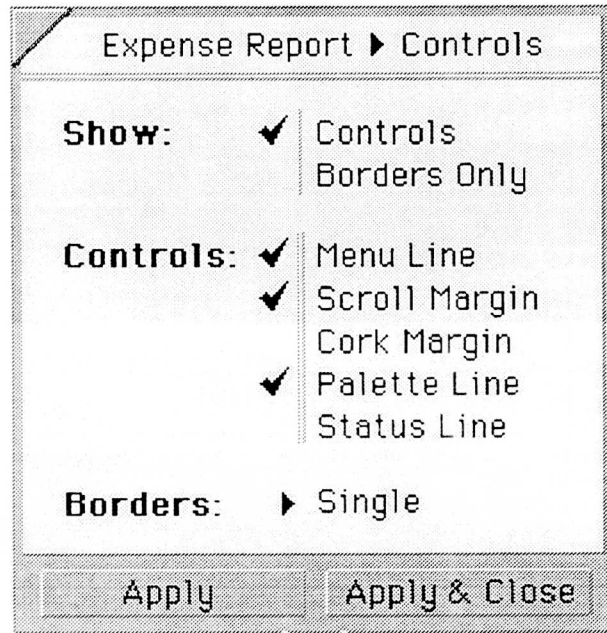
▼ Adding to the Default Document-Wide Option Sheets

The PenPoint Application Framework implements three default option sheets: the **Controls** sheet for showing and hiding controls, the **Access** sheet for controlling the user's access to the document, and the **Comments** sheet for basic attributes of the document.

The sections that follow give examples of how to customize these three option sheets.

Controls Sheet

In the example that follows, the application has added **Palette Line** and **Status Line** to the list of controls that the user can show or hide:



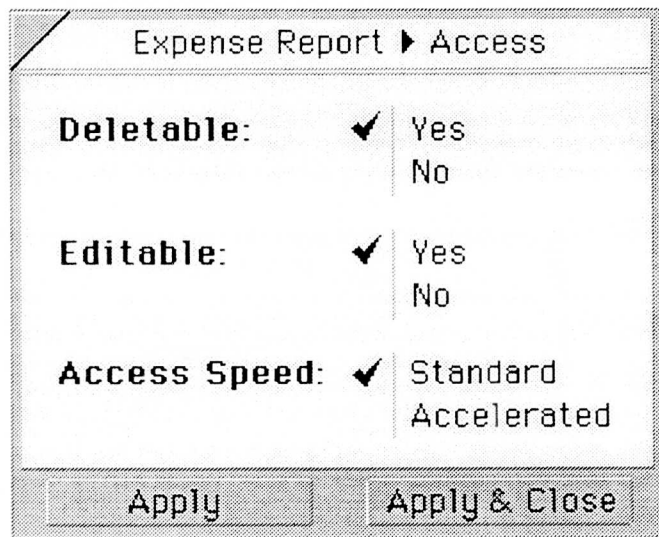
The dialog box is titled "Expense Report ▶ Controls". It contains three sections: "Show:", "Controls:", and "Borders:". The "Show:" section has a checked checkbox next to "Controls" and an unchecked checkbox next to "Borders Only". The "Controls:" section has checked checkboxes next to "Menu Line", "Scroll Margin", "Cork Margin", "Palette Line", and "Status Line". The "Borders:" section has a dropdown menu set to "Single". At the bottom are "Apply" and "Apply & Close" buttons.

Section	Item	State
Show:	Controls	Checked
	Borders Only	Unchecked
Controls:	Menu Line	Checked
	Scroll Margin	Checked
	Cork Margin	Checked
	Palette Line	Checked
	Status Line	Checked
Borders:	Single	Selected

Customized Controls Sheet

Access Sheet

The example below shows the Access sheet with an Editable control added:



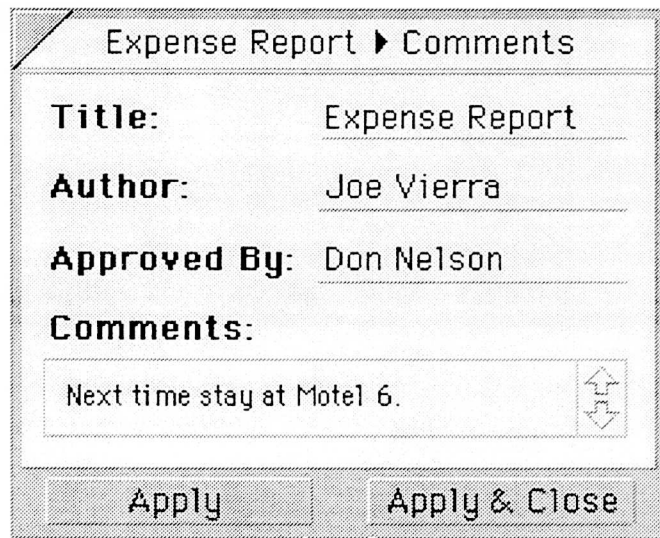
The dialog box is titled "Expense Report ▶ Access". It contains three sections: "Deletable:", "Editable:", and "Access Speed:". The "Deletable:" section has a checked checkbox next to "Yes" and an unchecked checkbox next to "No". The "Editable:" section has a checked checkbox next to "Yes" and an unchecked checkbox next to "No". The "Access Speed:" section has a checked checkbox next to "Standard" and an unchecked checkbox next to "Accelerated". At the bottom are "Apply" and "Apply & Close" buttons.

Section	Item	State
Deletable:	Yes	Checked
	No	Unchecked
Editable:	Yes	Checked
	No	Unchecked
Access Speed:	Standard	Checked
	Accelerated	Unchecked

Customized Access Sheet

Comments Sheet

In the example below, the application has added an **Approved By** field to the default **Comments** sheet:



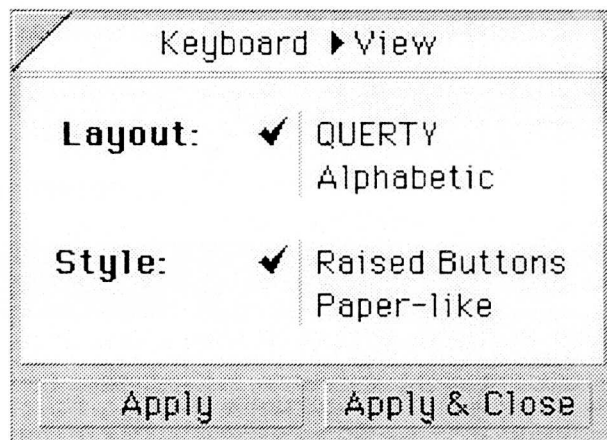
A dialog box titled "Expense Report ▶ Comments". It contains four labeled fields: "Title:" with the value "Expense Report", "Author:" with the value "Joe Vierra", "Approved By:" with the value "Don Nelson", and "Comments:" with the value "Next time stay at Motel 6." and a vertical scroll bar. At the bottom are two buttons: "Apply" and "Apply & Close".

Customized Comments Sheet

Adding a New Document-Wide Option Sheet

Add option sheets as needed for document-wide options that don't fit on the three default option sheets.

For example, suppose you had a pop-up keyboard accessory and wanted to provide several layouts of keys, and also allow the user to choose either a "realistic" look with raised keycaps on a grey background, or a more compact, lighter, "paper-like" look. The place to put the controls would be on an option sheet like the following figure. Any document-wide option sheets that you add should appear on the **Options** menu *before* the standard option sheets.



An option sheet titled "Keyboard ▶ View". It contains two sections: "Layout:" with a checked checkbox and the options "QUERTY" and "Alphabetic", and "Style:" with a checked checkbox and the options "Raised Buttons" and "Paper-like". At the bottom are two buttons: "Apply" and "Apply & Close".

Option Sheet for a Keyboard Accessory

Dynamic Behavior of Modeless Option Sheets

This section describes the dynamic behavior of option sheets—how they respond as the user makes a check gesture over an object or taps to change the selection.

It is very important to follow consistently the conventions described below whenever you use option sheets. The PenPoint User Interface Toolkit includes a message protocol to facilitate consistent implementation by applications.

Response to Check Gesture

When the user draws the check gesture over an object that is not selected, your application should first select the object that was the target of the gesture, then display its option sheet. It is essential to first select the object, so that the user can clearly tell which object is associated with the sheet.

This implies that you need to decide what the most natural and useful “default” object is in the context of your application. For example, in the table of contents, the default object is the entry (not the word) under the gesture, while in text the default object is the word (not the character) under the gesture.

If an option sheet is already up when the user draws a check on another object of the type appropriate to the option sheet, your application should first select the object, then update the sheet’s contents to reflect the object’s current state.

Relationship to the Selection

When there is no current selection, or the current selection is not of the option sheet’s type, you need to deactivate the **Apply** and **Apply & Close** buttons, because there is no target for the actions.

Note that selecting another object while an option sheet is open does *not* change the values for the controls on the option sheet.

If the user selects another object (of the option sheet’s type) while an option sheet is displayed, the PenPoint User Interface Toolkit automatically marks all the controls on the sheet as dirty. This allows the user to easily copy properties from one object to another by simply bringing up the option sheet for the source object, selecting the destination object, and tapping **Apply**.

When to Change the Top Sheet

If your application presents objects that appear to the user as clearly distinct types (text, figures, and bitmaps are typical examples), it is usually best to present a separate option sheet for each type of object.

Note that there is not necessarily a one-to-one relationship between option sheets and types of objects—a given sheet can be applicable to more than one type of object, and vice versa. It's up to you to specify which sheets are applicable to which types of objects.

If you have more than one sheet for a given type of object, you need to decide: 1) which is the primary sheet, and 2) when to change top sheets.

MiniText provides a good example. There are four sheets: **Character**, **Paragraph**, **Tab Stops**, and **View**. The first applies to characters, the second to paragraphs, and the remaining two to the document as a whole.

Because text has a hierarchical structure, when the user makes a check gesture on a word, he or she is also checking on a paragraph. MiniText defines the **Character** sheet as the primary one, so if the user makes a check gesture on a word, the **Character** sheet comes up. However, because the **Paragraph** sheet is also applicable, if the option sheet is up and the paragraph sheet is on top, when the user makes a check gesture over another paragraph the paragraph sheet remains up—it is *not* replaced by the **Character** sheet. This makes it convenient for the user to go from paragraph to paragraph, viewing and modifying either character or paragraph options.

Global Application Options

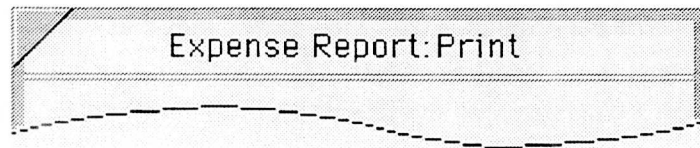
If you have options that apply globally to all existing or future instances of your application, add a sheet to the **Options** menu for each document. Put the sheet at the bottom of the menu, separated by a line from the **Comments** sheet. That way the order of the sheets is maintained: most immediate scope (the current selection), followed by the document as a whole, followed by all instances of the document.

Dialog Sheets

Use dialog sheets to allow the user to specify the parameters to commands.

Single and Multiple Dialog Sheets

If the dialog sheet is single, put the name of the document at the left of the title line, then the name of the sheet, separated by a colon:



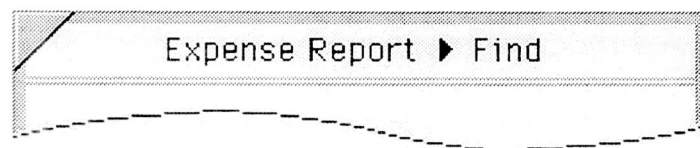
Title for a Single Dialog Sheet

Keep each sheet simple and compact—six to ten controls to a sheet is a good limit. The larger the sheet, the more of the screen it will obscure when displayed. One of the advantages of modeless sheets is that the user can leave them displayed and use them repeatedly. This feature becomes less useful the larger the sheet becomes.

To keep dialog sheets simple and compact, group the controls into separate categories, each with its own sheet. Examples from the standard PenPoint commands include **Print Setup**, which has three sheets, and **Find**, which has two.

When you create multiple sheets, the PenPoint User Interface Toolkit automatically generates a pop-up list containing the titles of all the sheets, and places this control in the sheet's title line. The arrow in the title line gives the user a consistent visual cue that there are multiple sheets.

To complete the title, all you need to supply is the name of the document. Omit the colon for multiple sheet titles, since the pop-up list arrow acts as a separator:



Title for a Multiple Dialog Sheet

Modeless and Modal Dialog Sheets

By default, PenPoint dialog sheets are modeless. You can also specify that a sheet be modal, in which case pen input to the rest of the screen is blocked as long as the sheet is displayed.

In most cases it is better to use modeless sheets, because they are less restrictive to the user. For example, if the user brings up the **Print** sheet and then wants to make a last-minute modification to the document before printing, there's no need to first close the sheet and then bring it up again.

Modal sheets are appropriate in two types of circumstances:

- ◆ When the user has initiated a command that displays a series of dialog sheets to lead the user through each step of a multi-step process. In that case, allowing the user to interrupt the process, do something else, then resume, may have little benefit and serve only to open the door to user confusion.
- ◆ When there is a need to restrict input to the sheet itself. For example, suppose a database application presented a dialog sheet to modify the result of a database query. It's quite possible that a modeless sheet would allow the user to change the state of the program's data such that it would be impossible or impractical to update the contents of the sheet. In such a case, using a modal sheet can eliminate a whole class of problems.

✦ **Layout Issues**

This section gives guidelines for laying out the controls in both dialog sheets and option sheets.

✦ **Standard Layout**

The default layout supported by the PenPoint User Interface Toolkit is designed to be simple and readable. It has two columns: The labels are on the left, the values are on the right.

The labels are shown in bold, followed by colons.

Both columns are left-justified. The text in the value column lines up vertically, whether the control is a choice list, a pop-up list, or a text field.

All parts of each control—both label and value—are aligned on the same horizontal baseline.

✦ **Non-Standard Layouts**

First try to design your sheets within the standard two-column layout. If the standard layout is too restrictive for your purposes, the PenPoint User Interface Toolkit will support more complex layouts.

If you find that you need to depart from the standard layout, remember that it is very important that the user be able to scan the sheet quickly to find the control of interest. The most important way to ensure this is to align all the controls within a clear (implied) grid. By following this basic rule of graphic design, you can maintain readability even in quite dense layouts. Failure to maintain a clear grid, on the other hand, will degrade usability.

Pop-Up versus In-Line Choices

As described in Chapter 6, the PenPoint User Interface Toolkit provides two forms of choice lists: an in-line form in which all the choices are shown, and a more compact pop-up form in which only the current choice is shown. Here are some guidelines for choosing which to use in dialog and option sheets.

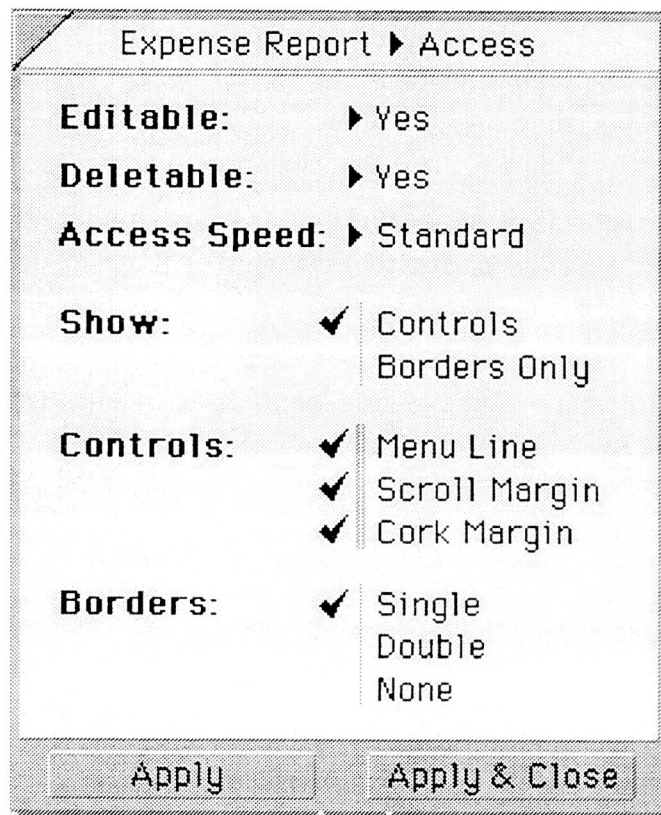
Using only one form of list on a single sheet has the advantage of simplicity: The sheet looks more orderly, and it's simpler for the user to only have to deal with one type of control.

First consider using the in-line form, which is easier to browse and use, for all of the lists on the sheet. If that would make the sheet too tall, then consider using the pop-up form for all the lists.

This is not a hard and fast rule. There are two good reasons to mix in-line and pop-up lists on a single sheet:

- ◆ If the controls fall clearly into frequently used and seldom-used controls, consider presenting the frequently used lists in-line and the seldom-used lists as pop-ups.
- ◆ If there is a natural grouping of controls, consider showing the related controls in the same form.

Here's a sheet that provides a good example of both guidelines:



The image shows a dialog box titled "Expense Report ▶ Access". It contains several settings, each with a label and a list of options. The first three settings ("Editable:", "Deletable:", and "Access Speed:") use a triangle symbol (▶) to indicate a pop-up list. The last two settings ("Show:" and "Borders:") use a checkmark symbol (✓) to indicate an in-line list. The "Show:" list has two options: "Controls" and "Borders Only". The "Borders:" list has three options: "Single", "Double", and "None". At the bottom of the dialog box are two buttons: "Apply" and "Apply & Close".

Label	Symbol	Options
Editable:	▶	Yes
Deletable:	▶	Yes
Access Speed:	▶	Standard
Show:	✓	Controls Borders Only
Controls:	✓	Menu Line Scroll Margin Cork Margin
Borders:	✓	Single Double None

Buttons: Apply, Apply & Close

Using Pop-Up and
In-Line Lists to Group
Controls

The guideline of using the same type of list control throughout the sheet would suggest that the **Show** and **Borders** also be shown as a pop-up list in the above example, like the first three lists on the sheet. But presenting them as in-line lists, like the multiple checklist to which they are related, helps the user infer the relationships between the controls. It's clear at a glance that there are two groups of controls: the triangle group at the top, and the check group at the bottom.

Also, in this case the **Show** and **Borders** controls are likely to be used more frequently than the three pop-up lists, so presenting them in-line makes sense.

Deactivating versus Hiding Controls

As with menus, when a control is not available for any reason, you should let the user know by deactivating it.

Often a control is active or inactive depending on the state of the control above it. The figure on the previous page contains such a pair. The **Show** toggle controls whether the **Controls** list or the **Borders** list applies. In the case of such related controls, activate and deactivate the dependent control immediately as the dominant control is changed, rather than only after the **Apply** button is tapped. This helps to make the semantic relationship between the two controls clear to the user.

It's also OK to hide a dependent control completely when it is irrelevant. If you do so, put the control that comes and goes at the bottom of the sheet, if possible, so that it doesn't leave a gap in the middle of the sheet.

With deactivating and hiding, as with other stylistic choices, it is best to choose one method or the other and stick to it throughout your application.

Chapter 9 / Status, Warning, and Error Feedback

This chapter gives guidelines for giving feedback to the user, including:

- ◆ Wording guidelines for messages and button labels.
- ◆ Progress and completion messages.
- ◆ Confirmation notes.
- ◆ Error notes.
- ◆ Timing-triggered notes.
- ◆ Audible feedback for warning and errors.
- ◆ Message lines.

► Wording Guidelines

This section gives guidelines for the wording of the messages and button labels in notes.

► Wording of Messages

- ◆ Use normal mixed case for the message. Don't use capital letters for emphasis.
- ◆ Don't use exclamation points. They give an exaggerated, cartoon-like impression. And they're not really necessary!
- ◆ Never use words that may have offensive connotations, such as "abort" or "kill."
- ◆ Put the message in the user's terms, not the programmer's terms. For example, "Not enough memory for operation. Press the Restart button, then try again" is more helpful than "Heap allocator: unable to malloc 4 K block."

► Wording of Button Labels

The PenPoint™ operating system supports modal notes for the display of progress, completion, confirmation, and error messages, as described later in this chapter.

In labeling buttons on modal notes, follow the convention described in the preceding chapter for modal dialog sheets: the name of the command, and **Cancel**. For modeless notes, use **Close** instead of **Cancel**.

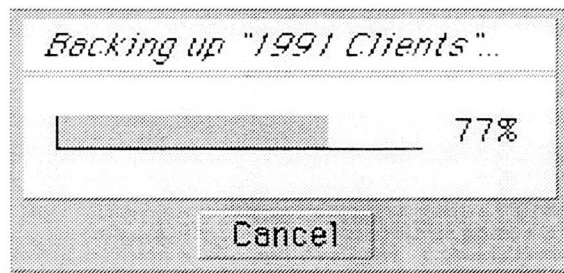
Using active verbs makes the meaning of the buttons unambiguous. It also allows the buttons to stand on their own, so the user can interpret them at a glance, without needing to read the message carefully.

Don't label buttons Yes and No. That forces the user to read the labels carefully to know how to interpret them—does Yes mean “Yes, proceed with the operation” or “Yes, I want to cancel the operation?”

Progress and Completion Messages

If an operation is very lengthy or especially important to the user for any reason, it's a good idea to supplement the standard busy clock that PenPoint provides with an explicit message while the operation is in progress.

The note shown below tracks the progress of a disk formatting operation:

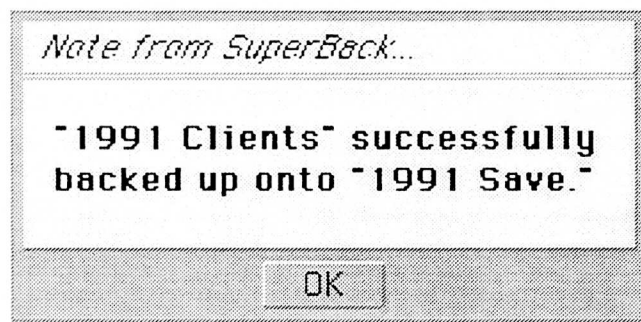


Progress Note

Whenever feasible, put a Cancel button on such progress notes to allow the user to cancel long operations.

Ordinarily, you should *not* force the user to explicitly acknowledge the normal completion of an operation. Simply take down the note when the operation is done.

If it *is* important that the user acknowledge the completion of the operation, leave the note up, change the message to reflect the completion of the operation, and change the label on the button from Cancel to either OK or Continue:



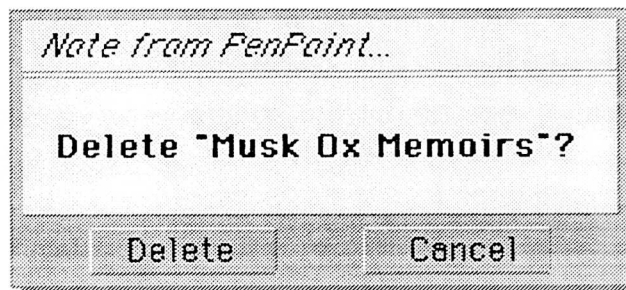
Completion Note for Lengthy Operation

This is a good practice for lengthy operations during which users are likely to go away and do something else. When they return, the completion message serves to reassure them (and perhaps remind them of what they were doing).

Confirmation Notes

Before proceeding with any operation that destroys data and cannot be undone, you should always protect the user by asking for confirmation.

The figure below shows a typical confirmation note:

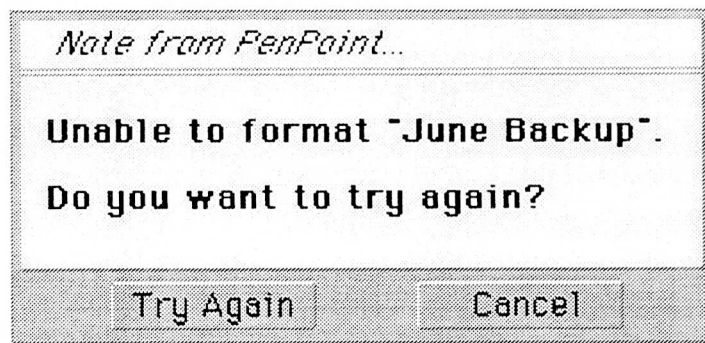


Confirmation Note

As with all notes, use active verbs for button labels, rather than Yes and No.

Error Notes

The figure below shows a typical pop-up error note:



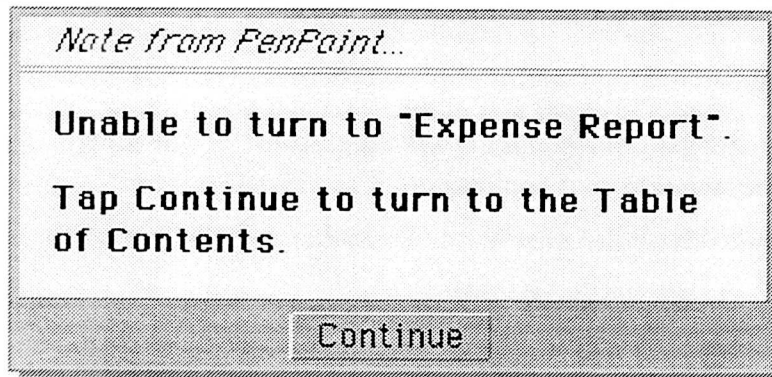
Error Note

Whenever possible, tell the user what he or she can do next. In the above example, the options are to try the operation again or to cancel the operation.

Timing-Triggered Notes

In some situations, it is appropriate to display a note for a few seconds, then automatically dismiss it and continue. You can use this technique both for completion notes and error notes.

For example, when the user tries to turn to a notebook page, and the PenPoint operating system is unable to open the document on that page, it displays this note:



Timing-Triggered Error Note

If the user doesn't tap Continue within five seconds, PenPoint takes down the note and turns back to the table of contents.

Use timing-triggered notes with caution. Never use them in situations where it is really important that the user see the note.

An inherent problem with timing-triggered notes is how long a timeout interval to use. Dismiss the note too quickly and the user will strain to see it, or miss it entirely. Leave it up too long and the user may become impatient waiting for it to go away. Given that different users will vary in how fast they read, it's not clear that one interval will work for the range of users that will use your application.

If you do use timing-triggered notes, it is especially important to test the interval on users.

▼ Audible Feedback for Warning and Errors

The PenPoint operating system is designed to be used anywhere—in meetings, during interviews, in classrooms, and so on—without intruding into the social situation.

For this reason, PenPoint provides a user preference for audible feedback, which by default is disabled.

You can use audible feedback for minor errors, subject to these two guidelines:

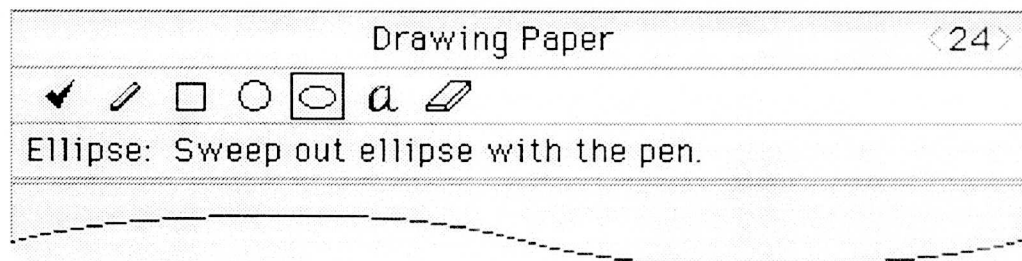
- ◆ Always determine the user preference before providing audible feedback. Never provide audible feedback when the user has indicated that it is not desired.
- ◆ Never rely solely on audible feedback, since it might be turned off by the user. Always accompany the audible feedback with some kind of visual feedback (such as briefly flashing the title line).

If the user has enabled the audible feedback preference, notes will be accompanied by audible feedback when they are displayed. You can override this behavior so that a particular note is never accompanied by audible feedback, regardless of the preference setting.

▼ **Message Lines**

If you need to give many messages or instructions to the user, consider providing a message line as part of the layout of your application.

For example, an application with a palette line for different drawing modes might have a message line giving instructions for each mode:



Message Line

Note that the message line is on the top rather than the bottom of the application's work area. Because of the way the PenPoint computer is held in the hands, the top half of the screen is more easily seen than the lower half, which is more often obscured by the user's hand and forearm.

If you do put a message line at the bottom, make sure that you do adequate user testing to verify that users see the messages.

One of the considerations underlying the use of a message line is the basic issue of how much feedback to give to the user. Too little and the program may be seen as terse and unfriendly. Too much feedback will get in the user's way, and may be seen as cumbersome or even condescending. The only way to be confident that you have the right balance is through user testing.

As with any extra line, put a control on the Controls sheet to allow the user to turn the message line on and off.

Chapter 10 / Help

This chapter gives guidelines for using the two facilities that the PenPoint™ operating system provides for presenting on-line application help: Quick Help and the Help notebook.

Quick Help

The purpose of Quick Help is to provide brief, context-specific help for anything the user can tap on.

The perceived value of Quick Help depends on complete coverage and accurate messages. Users are likely to become frustrated if they go to the trouble of bringing up the Quick Help sheet only to find that there is no message available. Unless users can count on complete, high-quality Quick Help, they are unlikely to use it more than once.

Provide Quick Help for all distinct regions of your application's display, including:

- ◆ All controls on option and dialog sheets, and the background and title lines of the sheet.
- ◆ Regions in the work area of your application. For example, a table should typically provide Quick Help for each column, and for the row/column border areas as well as for individual cells.
- ◆ Confirmation, warning, and error notes when the message is not completely self-explanatory.

Follow these guidelines when writing Quick Help messages:

- ◆ Keep the message brief. Ideally, all the text should fit in the default sheet without requiring scrolling.
- ◆ Mention any relevant gesture accelerators.
- ◆ Cross reference to a topic in the Help notebook if appropriate.

Help Notebook

While the Quick Help sheet is for very brief messages describing objects that the user can see on the screen, the Help notebook is the place to describe how to use your application.

As part of the installation process, you can specify that one or more documents be placed in Help notebook. These documents can be of any type.

Chapter 11 / Designing Application Icons

Whether you structure your application as a document, accessory, or notebook, you need to design an icon to represent it.

This chapter presents guidelines for designing icons. Following these guidelines will ensure that your icons fit in with those designed by GO and by other application vendors, thereby helping to bring visual consistency to the multi-vendor PenPoint™ operating system environment.

This chapter also describes the use of icons to show application state.

► GO's Conventions for Icons

This section describes the basic theme GO follows for its icons. You can follow it or not; in any case, having a sense of what your icon's neighbors will look like will be helpful as you go about the process of designing your icons.

► Icons for Documents

The figure below shows some typical document icons as they appear in the notebook table of contents:

	To Do	2	Document Icons in Table of Contents (12 pt.)
	Letter to Gertrude	3	
	Contract	4	
	Icon In Progress	5	

The icons are for MiniNote, MiniText, GO Fax application, and the PenPoint bitmap editor.

Note that the icons are all based on the outline of a document, and have some embellishment to identify the type of information contained in the document.

It's not important that you follow this strict convention, but by all means design your icon to fit in with the document and notebook metaphor.

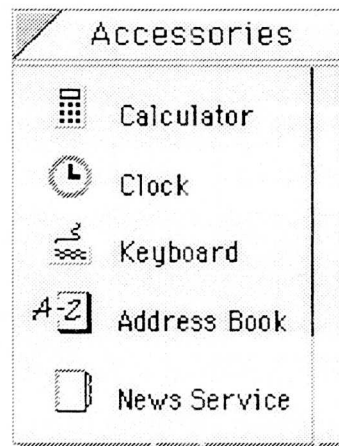
When it makes sense, the large icon has a pen. If the user scaled the system to use a 20-point font, the table of contents would look like this:

	To Do	2
	Letter to Gertrude	3
	Contract	4
	Icon In Progress	5

Document Icons in Table
of Contents
(20 pt.)

Icons for Accessories

The figure below shows some typical icons as they appear in the Accessories palette:



Accessory Icons

Note that these icons are *not* based on the standard document form. Instead, each suggests a familiar object closely related to the accessory's function. The News Service accessory in the above example is structured as a notebook, therefore it has a notebook-like icon.

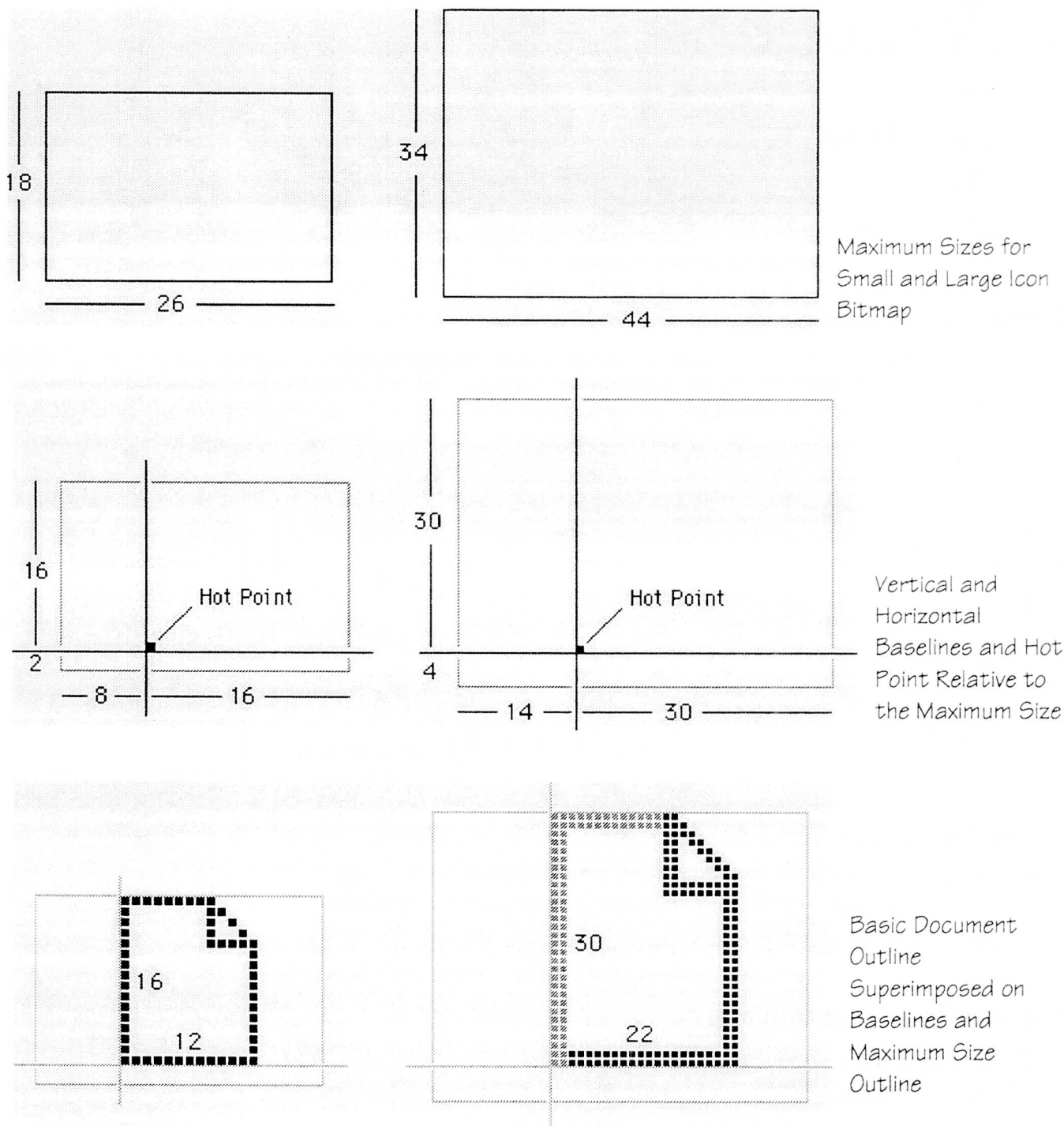
Icon Dimensions and Hot Point

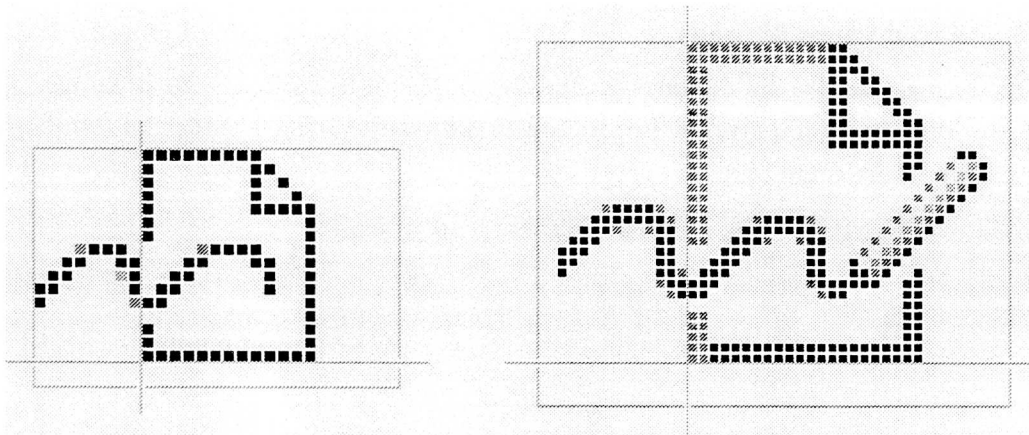
The icon option sheet allows the user to choose between two icon sizes: The default size is 16 pixels high, the large size 30 pixels high. You need to provide two bitmaps for your application's icon, one for each size.

When you make the bitmap, you also specify the **hot point** for the icon. It is essential to put the hot point in the right place, because the PenPoint operating

system uses it to line up the icons in the table of contents and in other contexts where icons are laid out in a column. If its hot point is in the wrong place, your icon will not line up correctly.

The figures that follow give the size limits and recommended hot point placement for both the small and the large bitmap:





Use the Area
Outside the
Document Outline
for Ornamentation

Icon Design Guidelines

This section gives guidelines for the graphic design of PenPoint icons.

Start with the Small Size

Do the small bitmap first. It is much easier to embellish a design for a larger bitmap than to simplify a design for a smaller bitmap.

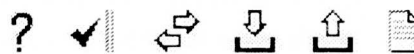
Keep It Simple

The most important rule is: *Keep the picture simple*. Don't try to capture too much detail in such a small space.

Given the limited the number of pixels in the small bitmap, a literal approach to designing your icon is unlikely to work well. Think about abstracting a single key element that can stand for your application and emphasizing that element visually. Bear in mind that the user will glance at your icon, not study it.

Remember also that the user will actually see your icon on a low-power computer screen that will almost certainly have lower contrast than paper. Detail gets lost.

The figure below, showing two versions of the same icons, illustrates how simple shapes communicate more directly:



Keep Icons Simple

Attempting too much detail makes the icons in the top row look busy.

Eliminating unnecessary detail makes the versions on the bottom easier to recognize.

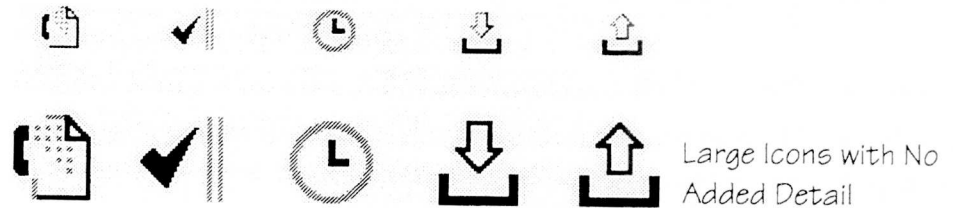
✦ Relationship of the Small and Large Bitmaps

When designing the large bitmap, don't take the extra pixels as an opportunity to redesign the icon. The two bitmaps should give the impression that the large one is basically a scaled up version of the small one.

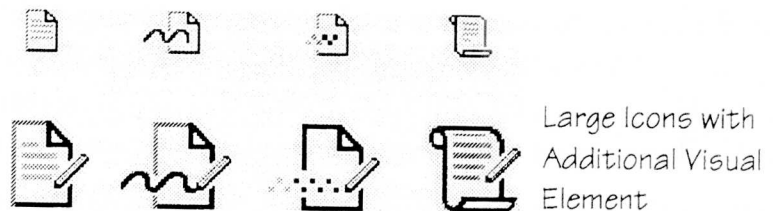
It's OK to add some detail to the large bitmap, as shown in these pairs of icons:



It's also fine to simply scale the picture up, without adding any detail at all:

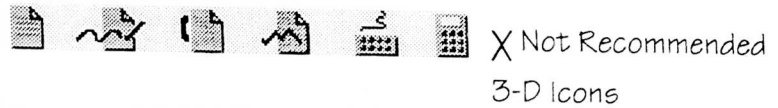


If you do add another visual element to the large bitmap, make sure that the additional element follows naturally from the design. In the icon pairs below, while the pen is only present in the large bitmaps, it is implied in the small ones:



⚡ Don't Use the "3-D" Style

It may be tempting to use the "3-D" style (light grey with dark accents suggesting shadows) for icons:

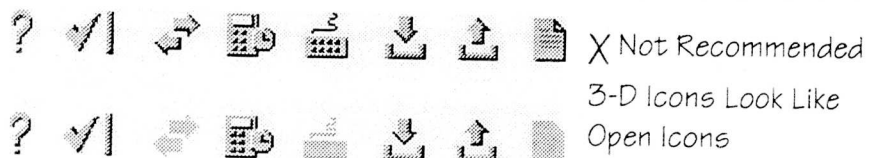


The 3-D style should be avoided, however, for three reasons.

First, remember that many PenPoint computers will have pocket-sized displays, and the user will be looking at the small icons, not the large ones. The small patches of light grey simply aren't clearly enough visible on the screens used for mobile, pen-based computers (which may not be back-lit). They may look fine on paper, but on real screens they tend to blur into fuzzy shapes that all look alike.

Second, while the 3-D style makes visual sense for things like keyboards and calculators, it is not as appropriate for sheets of paper, which are flat.

The final reason to avoid the 3-D look is that it is too close to the solid light grey used for the icon in its open state. Quick, which of the icons below is open?



⚡ Designing the Icon Mask

When you create the icon bitmap using PenPoint's bitmap editor, you specify two parts:

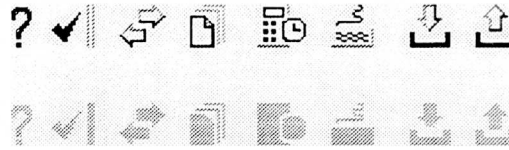
- ◆ The picture itself, in which each pixel can be either white, light grey, dark grey, or black.
- ◆ A shape, known as the **mask**, through which the icon will be painted.

The mask has no grey levels; each pixel is either on or off. The mask must be a superset of the picture; in other words, a given pixel in the picture will appear only if the corresponding pixel in the mask is on.

When the icon is in its normal (closed) state, the PenPoint operating system paints each pixel within the mask in the color that was specified when the icon was created. When the icon is in its open state, PenPoint paints each pixel within the mask in light grey. This allows the user to tell at a glance when an icon is open.

You need to design both the picture and the mask carefully to make sure that the user can distinguish between the closed and open states of the icon.

The figure below shows the default Bookshelf icons in both closed and open states:



Closed and Open Icons

Notice that the shapes that work best are the ones that are partially open and partially enclosed, so that there is some difference between the open and closed forms, and the distinctive shape of the icon can still be distinguished when the icon is open.

Also, notice that the masks are more filled-in and blockier than is strictly necessary. This makes the open icons more clearly visible, because light grey lines and dots tend to disappear.

▼ Using Icons to Show Application State

You can give the user useful feedback by changing the icon's picture to reflect the state of your application.

The figure below shows how the In box and Out box make use of this technique:



*Empty and Full In Box
and Out Box*

The icons at left indicate that the In box and Out box are both empty; the icons on the right indicate that In box and Out box each contain at least one document.

Chapter 12 / Putting the Building Blocks Together

The preceding chapters have introduced the various building blocks provided by the PenPoint User Interface Toolkit: controls, menus, dialog and option sheets, and so on.

With these building blocks in hand, we can now step back and ask how they should be put together. Should a given function go on a menu, an option sheet, a palette, or all three? Should all functions be put on menus?

This chapter gives principles and examples to help you answer the basic question of how to present your application's functionality.

Topics include:

- ◆ Dual command path: controls and gestures.
- ◆ Layering functionality to hide complexity.
- ◆ Allowing the user to configure the interface to your application.
- ◆ When to use menus, option sheets, and palette lines.
- ◆ When to depart from the standard building blocks.

▼ Basic Guidelines

This section mentions the most basic guidelines to keep in mind when thinking about how to present your application's functionality.

▼ Use of Gestures

To summarize the gesture guidelines:

- ◆ **Dual command path.** Throughout your design and development process, ask yourself how you can give your users the choice of using either gestures or visible controls such as buttons, checklists, menus, or palettes.
- ◆ **Core Gestures.** Use the core gestures whenever they are relevant to your application. Always follow the standard PenPoint usage for the core gestures.
- ◆ **Other Gestures.** Use the non-core gestures and the capital letters as accelerators for frequently used operations. Use gesture accelerators judiciously. The point isn't to find a gesture accelerator for every command, but to use them where they really make the user's job easier.

▼ Layering

This principle applies in designing any application, particularly those that present a rich set of functionality and will be used by a diverse set of people.

Don't overwhelm the user by presenting all of the commands at the same level. Layer the interface to hide complexity. Present the most important commands prominently, and put the rest underneath the surface, on menus or option sheets that the user must open to see.

▼ User Configurability

Think of how you can help the user by allowing him or her to configure the face presented by your application.

There are many ways to do this: by providing "novice" and "expert" modes, by letting the user put favorite commands on customized palettes, and so on.

One degree of flexibility you should always provide is to allow the user to hide any non-essential palette line, message line, or other region to make more room for the display of data.

Add the controls for showing and hiding the non-essential regions to the standard document Controls sheet, as illustrated in Chapter 8.

Where to Put It: Menu, Option Sheet, or Palette Line?

Now we come to an issue that is very basic, and yet difficult to give hard and fast guidelines for—how to divide your application’s functionality between menus, option sheets, and palette lines (lines above, below, or at the edge of the document’s work area that contain frequently used controls).

Strengths and Weaknesses of Each Mechanism

No single user interface mechanism is “best.” Each has its strength and corresponding weakness, as summarized in the following table.

Menus, Option Sheets, and Palette Lines

Mechanism	Strengths	Weaknesses
Menus	Menus invite the user to explore by tapping. Menus are easy to invoke for a single operation.	Menus are inefficient for frequently used commands, or for commands that are typically issued in groups, because they disappear automatically as soon as one command is issued.
Option Sheets	Option sheets don’t require the user to access each control by name. All the user needs to remember is to make the check gesture over the object of interest. Because they are modeless, option sheets allow the user to easily change several settings at once, or to keep the sheet up and use it repeatedly. Option sheets make it very easy to copy all the option settings from one object onto another.	Option sheets take longer to display than menus, and usually take up more screen real estate than menus (although the user can resize the sheet to show only the desired control). While option sheets are displayed, they obscure part of the screen, often requiring the user to drag them out of the way.
Palette Lines	Palette lines provide the most visual invitation, and the most efficient access, because all the controls are visible at once. Palette lines are typically compact, because they use small pictures instead of longer textual labels. Palette lines don’t obscure any of the document’s work area, and never need to be dragged out of the way.	Palette lines are not appropriate for commands that can’t be clearly expressed as a small picture. Palette lines are not appropriate for less frequently used commands. Palette lines take screen real estate from the document’s work area.

✦ How to Decide Which Mechanism to Use

When considering how to present your application's commands and options to the user, there are two useful questions to ask.

1 What are the most important, obvious objects that the user will see and deal with?

In general, you should provide an option sheet for each type of object in your application unless there's a compelling reason not to.

Whatever the object is—a figure in a graphics editor, a cell in a spreadsheet, a span of text, or an appointment entered into a calendar—the user should be able to view and modify the object's options with the check gesture, and edit the object's label (if there is one) with the circle gesture.

For example, suppose your application presents a list of sales contacts. Each contact in the list has associated information that the user can modify. You could call the command by a specialized name, such as **Update Contact Information**.

But that approach forces the user to learn and remember the specialized name you have chosen. By presenting the same command under the generic umbrella of **Options**, you lessen the learning curve. The user comes to your application already knowing about option sheets and how to invoke them. There's no need to even think in terms of a new command. The user simply makes a check gesture on the entry in the list, and already he or she is successfully using the application.

2 Is there a set of commands or modes that are fundamental to your application that the user will use frequently?

If the answer is "yes," it is probably best to present the commands in a palette that is always visible so the user never has to worry about popping up a menu or option sheet, or dragging an option sheet out of the way.

The above two guidelines will suggest what you should put on option sheets and palette lines; now, what about menus?

When designing your menu line, there are two general approaches you can take.

One is to put all the important functionality on the menu line so that the menu line, as the primary way of issuing commands, becomes indispensable.

The other is to put less functionality on the menu line and more on other input paths: gestures, palette lines, and option sheets. That way, the user can turn off the menu line (to reclaim the screen space it occupies) and continue using the application. In the notebook table of contents, for example, the user can turn off the menu line and browse through the notebook contents, expand and collapse sections, turn to documents, turn tabs on and off for specific documents, move, copy, or export documents, and so on.

▼ When to Depart from the Standard Building Blocks

Don't depart gratuitously from the standard conventions for controls, menus, option sheets, and the like. There's not much added value in being different for difference's sake, and there may well be a cost if it requires the user to learn new variations.

Take scroll margins, for example. Every application developer could customize the scroll margin visuals to achieve a "signature" look, or to add buttons to optimize various operations. But it's not clear that that would add significant value to any of the applications, and the net effect would certainly be to give the user more to learn. It's hard to imagine that customizing scroll margins would be the best use of the limited resources available to each developer.

But of course there are many situations in which it *is* appropriate to depart from convention. Guidelines and standards are not meant to be rigidly applied, or to produce a monotonous scene. In the oft-quoted words of Ralph Waldo Emerson, "A foolish consistency is the hobgoblin of small minds."

The best reason to depart from the standard visuals is to emulate a real-world object, such as a calculator, control panel, rolodex, or telephone message slip. In such cases, by all means be realistic.

In fact, the value of such an approach is that the realistically rendered object reduces learning time by taking advantage of the user's familiarity with similar objects from other contexts. So we remain grounded, after all, in the same old principle of consistency. The difference with realistic objects is that the assessment of consistency is with respect to things from other domains that the user is familiar with, instead of the domain of the standard PenPoint user interface.

Part 4

Processing Pen Input

DESIGNING A PENPOINT APPLICATION

PART 4 / PROCESSING PEN INPUT

▼ Chapter 13 / Processing Gestures	231		
Gesture Targeting	231		
The Hot Point	231		
Targeting Guidelines	232		
Passing on Unused Gestures	233		
Avoiding Capital Letter Gesture Collisions	233		
Gesture Families	235		
Usage Guidelines for Individual Gestures	236		
▼ Chapter 14 / Processing Handwriting	241		
Using Handwriting Translators	241		
Constraining Translation	241		
The Problem of Ambiguity	241		
Improving Translation with			
Context-Specific Constraints	242		
Constraining to Letters, Numbers,			
Punctuation, or Symbols	242		
Constraining with Dictionary			
and Templates	242		
Guidelines for Using Constraints	243		
▼ Chapter 15 / Presenting Input Modes	244		
The Role of Modes in PenPoint	244		
Spatial Modes	244		
Explicit Modes	244		
User Models	245		
Guideline Summary	246		
Core Gesture Mode	247		
Mode Switch	247		
Mode Toggle Gesture	247		
Mode Switch Presentation	248		
Pictures versus Text	248		
Dual Modes: Toggle Switch	248		
Multiple Exclusive Modes	248		
Non-Exclusive Modes	250		
Don't Hide the Mode Switch	250		
Mode Switch Location	251		
Menu Line	251		
Palette Line	252		
Pop-Up Palette	252		
Examples of Modal Applications	253		
Markup Layer	253		
Deferred and Immediate Translation	253		
▼ Chapter 16 / Using Ink	254		
		The Ink Data Object	254
		Ways of Using Ink	254
		Guidelines for Editing Ink	255
		Ink Mode	255
		Gesture Mode	257
		Gesture Margin	257
		Summary of Ink Gestures	258

Chapter 13 / Processing Gestures

This chapter covers the use of gestures, including:

- ◆ How to determine the operand of the operation represented by the gesture.
- ◆ What to do with gestures you don't use.
- ◆ Avoiding recognition collisions between gestures and capital letter accelerators.
- ◆ Guidelines for how to use each gesture recognized in PenPoint 1.0.

▼ Gesture Targeting

The object that the user intends to operate on is called the **target** of the gesture.

The problem of determining the intended target for a given gesture in a given context has two aspects.

First, the most natural or useful focal point of the gesture must be determined. The PenPoint™ operating system addresses this by defining a precise **hot point** for each gesture, as described in the next section.

The next step is for you to translate the hot point into a well-defined object in your application, based on the semantics of your application.

The remainder of this section describes hot points and gives guidelines for determining the target of the gesture.

⚡ The Hot Point

Gestures fall into a few groups based on their hot points:

- ◆ **Pen-down point.** For a large group of gestures, PenPoint treats the point where the pen first touches the screen (the **pen-down point**) as the hot point. Such pen-down gestures include the pigtail and the tap, press, flick, corner, and check families.
- ◆ **Center of bounding box.** For some gestures, the PenPoint operating system treats the center of the gesture's bounding box as the hot point. These gestures include cross-out, plus, the circle family, and all the capital letters.
- ◆ **Visual focal point.** When a shape has a strong visual focal point, PenPoint treats that point as the hot point. The gestures with the strongest visual focal point are the caret and arrow families.
- ◆ **Other.** A few gestures don't fit neatly into the above categories. The most useful hot point for the brackets is the midpoint of the bracket's edge (left edge for left bracket, right edge for right bracket). The most useful target

point for circle-flick-up and circle-flick-down is the pen-down point of the flick, within the circle. The undo gesture doesn't really have a hot point, in the sense that it doesn't act on a particular object.

Targeting Guidelines

This section gives guidelines for targeting both unselected objects and objects that are currently selected, and also discusses issues in targeting objects that are implemented as windows.

Targeting Unselected Objects

One of the pen's inherent strengths as an input device is that it allows the user to indicate both the operation and the operand in a single, natural gesture.

In order to take full advantage of this capability, allow the user wherever appropriate to gesture over objects without first selecting them.

To target without prior selection, you need to decide what, from the user's perspective, is the most intuitive or useful object to use as the default target for the operation.

Let's look at some examples from PenPoint's text component. A few gestures act on characters: tap to select, pigtail to delete. But most editing gestures—cross-out to delete, circle to edit, check to set options, circle-flick to find the next occurrence of a word, and so on—act on whole words, because the word is the most useful unit of text to edit.

Note that the default target for a given gesture may well differ according to the context. For example, a cross-out deletes a word when made in text, deletes a document or section when made in the notebook table of contents, and deletes the entire contents of the field when made in a fill-in field.

Targeting the Selection

To allow for the targeting of unselected objects, it is important that gestures act on the selection *only* if made over the selection.

If the selection is very small, it may be difficult for the user to target it accurately. To take an extreme example, a selected lower-case letter 'i' in a 10 point, variable-width font is typically only three pixels wide.

To facilitate the targeting of such small selections, allow a small "targeting zone" around the target point. If this region intersects the selection at all, take the selection as the target.

The PenPoint text component employs a targeting zone that is three pixels wide by five pixels high. This makes it easy enough, for example, to delete a single character with a pigtail.

☛ Targeting Window Objects

The input system distributes each gesture to the window that the pen first touches when making the gesture.

This presents an implementation problem when the gesture's target point is not the pen-down point, and the object to be targeted is implemented as a separate window.

For example, let's take the common case of making a cross-out to delete the contents of a field in a form. Since the idealized target point of the cross-out is the center of the bounding box, the user will typically start the cross-out slightly above the text to be deleted, and end it slightly below. However, if the field is implemented as a separate window, it will often never see the gesture, because the input system will distribute it to the window implementing the background of the form.

The PenPoint User Interface Toolkit addresses this by providing a window class that supports a more flexible targeting policy. You can specify that the parent window distributes the gesture to the first of its children windows to be intersected by the gesture.

☛ Passing on Unused Gestures

The general rule for processing gestures is to pass on to your ancestors all gestures that you aren't interested in.

This allows the ancestor to provide standard default behavior when appropriate. Take the caret family as an example. Caret, caret-tap, and double-caret all bring up the Create menu by default in class EmbeddedWin. The PenPoint text component displays a pop-up writing pad on caret, and an embedded writing pad on caret-tap. By passing the double-caret on, it gets the desired default behavior of displaying the Create menu.

☛ Avoiding Capital Letter Gesture Collisions

The term "gesture collision" refers to what happens when two gestures are not distinct enough to be reliably distinguished by the gesture recognition engine.

Gesture collisions are frustrating from the user's standpoint, because they cause the system to behave unpredictably. For example, suppose the bracket gesture to adjust a selection collided with the flick gesture for scrolling. The user would attempt to extend the selection only to find that it suddenly (for no good reason) hopped to the top or bottom of the page. Only by figuring out that the bracket had been mis-recognized as a flick could the user again make sense of the situation.

While collisions can never be completely eliminated, all of the core and non-core PenPoint gestures have been designed, tested, and tuned to minimize collisions.

However, adding the 26 capital letters to the overall gesture set introduces many potential collisions. Two are obvious and unavoidable: the letter O and the circle gesture are simply two ways of referring to the same shape; the same goes for the letter X and the cross-out gesture.

Collisions are not limited to these obvious cases. For example, the letter C often collides with the left bracket gesture. The underlying reason for collisions between gestures and capital letters is that the capital letters are recognized by a separate character recognition engine. Character recognition engines are typically tuned for broad coverage of handwriting, and include many shapes for each letter.

When using capital letter gestures, *it is your responsibility to make sure that the gestures you use don't collide with core or non-core gestures.*

The most important rule is: If you find that you want to use a letter that collides with a core gesture, the core gesture should always take precedence, because it is more widely useful throughout the system.

This rule applies as well to gestures that aren't part of the official list of 11 core gestures, but do have well-defined meanings in the PenPoint operating system, such as the brackets to adjust the selection, or up-down to toggle the borders of embedded documents on and off.

Let's look at an example. Suppose you want to use the letter C as an accelerator to create a new item in a list. But if the list supports multiple selection, it should support the left and right bracket gestures to adjust spans of selected items. Unfortunately, the left bracket collides with the letter C. Adding the C will cause unpredictable behavior, because mis-recognition will occur both ways: C's will be mis-recognized as brackets, and brackets as C's. The only solutions are to find another accelerator for the **Create** command, or to forego the accelerator entirely.

It may be appropriate in some cases to map two gestures onto one operation. Let's look again at the previous example. If the list did *not* support multiple selection, then the brackets would not be needed for the "adjust selection" function. But the potential for mis-recognition would remain high: Some percentage of reasonable C's would be mis-recognized as left brackets. The best solution would be to accept both left bracket and C as accelerators for **Create**. That way the user would be successful even if he or she made a C that was misrecognized as a left bracket.

The same rule applies to capital letters that collide with each other, such as U and V: never use such a pair for two different operations. If you do use either of the pair, accept the other letter as well, and map both to the same operation.

A final note of caution: Some collisions are easy to predict from looking at the form of the gesture, but handwriting recognition is a young and imprecise art, and often collisions cannot be predicted. The only way to be sure is to test your gesture set on real users.

Gesture Families

One of the keys to the learnability and usability of PenPoint's gestures is the use of gesture families.

A gesture family typically consists of a core gesture and several variations. By associating similar shapes for similar meanings, gesture families help the user learn and remember gestures.

The following table summarizes the gesture families for PenPoint 1.0.

Gesture Families

Gesture Family	Meaning	Comments
Taps	Select objects or invoke controls	Tap is the basic gesture for selecting objects and invoking controls such as buttons, menu commands, and checklists. Double-tap is used to float documents from icons, tabs, or reference buttons. In many situations, one, two, three, and four taps are used to select a hierarchy of objects: for example, character, word, sentence, and paragraph in text.
Presses	Move/Copy/Link/View Also used to sweep out a rectangle, usually for area select, occasionally for other purposes (for example, zooming)	The first family of meanings all has to do with viewing data from somewhere else. Press moves and tap-press copies throughout PenPoint. Double-tap press will probably be used in future releases of PenPoint to create a reference button linked to the point of the gesture. Use triple-tap-press to create a hot link, or alternate views of an object. Press is also used to sweep out a region for selection. For example, press begins a wipe-through selection when done over unselected text. Press in the margin of a list should also wipe-through select, and press on the background of a graphics editor should sweep out a bounding box for selection.
Flicks	Bring more information into view	Depending on the context, bringing more information into view means scrolling a sheet, turning a page, or cycling through a list.
Circles	Edit	Use circle for the primary meaning of edit in a given context, and circle-tap for the secondary variation. Circle-line means replace. Double-circle means create reference button throughout PenPoint (the double-circle shape suggests a link). Use circle-flick up/down to find the next/previous occurrence of a word in text.
Carets	Insert or Create	Use caret for the primary meaning of create or edit in a given context, and caret-tap for the secondary meaning. Don't process double-caret; PenPoint uses it to pop up the Create menu.
Checks	Options	Use check to display the object's option sheet. If needed, use check-tap for a variation related to options.

Gesture Families (Continued)

Corners	Many meanings	There are eight corner gestures. For most of the corners, there are corresponding corner-flick gestures: The user can make a flick in the direction of the second leg of the corner as an intensifier. For example, in text, down-right inserts a space, and down-right-flick inserts a tab; right-up means initial caps, right-up-flick means all caps. Sometimes it is useful to make use of the length of the second leg. In overwrite fields, down-right inserts one or more blank boxes, depending on the length of the right leg. In the ink component, right-down and left-down in the gesture margin open up white space, and right-up and left-up close up white space, according to the length of the vertical leg.
Arrows	Many meanings	There are four arrows and four double-shafted arrows.

Usage Guidelines for Individual Gestures

This section gives specific guidelines for the use of each gesture recognized in PenPoint 1.0.

For the purpose of presenting the guidelines, the gestures are grouped into four categories. From most to least rigidly specified, the categories are:

- ◆ **Core.** The 11 most important gestures that make up the core set that users learn first. These gestures must produce consistent results throughout the PenPoint operating system.
- ◆ **Well-defined.** Additional gestures for which PenPoint defines a meaning that is consistent across all applications.
- ◆ **Lesser members of core families.** Variations on the core gestures. The meanings of these gestures vary from application to application, but should always be related to the core gesture they derive from.
- ◆ **Open.** Gestures for which PenPoint does not define a meaning. These are available to use for application-specific meanings.

The pages that follow present four tables, one for each category of gesture.

Guidelines for Core Gestures

	<i>Tap</i>	Use for the most basic or important function of an object or region, typically selecting an object or activating a control.
	<i>Press</i>	When done over an object that can be moved by the user, press should always initiate a move of that object. The secondary meaning is to sweep out a region for selection. For example, when done in text over a selection, press begins a move for the selection; when done in text not over a selection, it begins a wipe-through selection. Press in the margin of a list should also wipe-through select, and press on the background of a graphics editor should sweep out a bounding box for selection. If your application supports area select, use press to sweep out the region to select, and use tap-press to sweep out a rectangle and perform an operation immediately. If your application doesn't support area select, you can also use press to sweep out a rectangle and perform an operation (such as zooming) in a single step.

Guidelines for Core Gestures (Continued)



Tap-Press

When done over an object that can be copied by the user, tap-press should always begin a copy of that object.

When there is nothing to copy, you can use tap-press to sweep out a rectangle for some purpose other than copying.



Flick
(four directions)

Use whenever there's a need to bring more information into view. The user model is that the flick gesture shoves the object in the direction of the flick.

PenPoint uses flicks to scroll text, turn notebook pages, expose overlapping tabs, and cycle through choices in pop-up checklists.

An application-specific usage might be to flick on a small monthly calendar to cycle to the next or previous month.



Cross Out

Use to delete the default object in your context.

Examples from PenPoint include: deleting a word in text, a document in the table of contents, and the entire field contents in a fill-in field.



Caret

Use for whatever the primary meaning of insert or create may be in your context.

Examples from PenPoint: Caret pops up a writing pad in text, pops up the **Create** menu in the table of contents, and inserts the date in the ink component.



Circle

Use for editing the primary attribute of an object, typically the object's textual label.

Examples from PenPoint include: entries in the table of contents, titles in the document title line, tab labels, icon labels, the entire contents of a fill-in field, a word in the text component, and an ink object in the ink component.

Typical application-specific objects would be a field in a form or a cell in a table.



Check

Use to display the option sheet for an object.

The object may be a document, an accessory, a span of text, an icon, a figure in a graphics editor, a cell in a spreadsheet, and so on. This gesture is a workhorse that can almost always carry the burden of changing whatever needs to be changed for an object or region.



Undo

Undo the last operation that can be undone.

It's up to you to define which operations in your application can be undone. Some applications only allow deletions to be undone; others allow almost all operations, even selection, to be undone.



Pigtail

Use to delete smaller objects (if there are any) than the default object deleted by cross-out.

In text, pigtail should delete a single character.



Down-Right

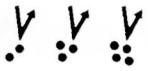
Use to insert a space in text.

In translated text, inserts a single character. In overwrite boxes, the number of spaces depends on the length of the horizontal leg of the L.

Guidelines for Well-Defined Gestures

[]	<i>Brackets</i>	Use to adjust an existing selection in any context that supports the selection of a span of objects (such as text, lists, and tables). The left bracket adjusts the starting point of the selection, the right bracket adjusts the end point.
+	<i>Plus</i>	Use to toggle selection in any contexts where discrete, discontinuous selections make sense, such as graphics editors or lists. It's also OK to use plus in its mathematical sense, to mean sum.
=	<i>Scratch Out</i>	Use to delete any object touched by the gesture. Examples from PenPoint include: In text, scratch out deletes words; in overwrite fields, it deletes characters; in ruled pads, it deletes individual strokes.
⊖	<i>Double-Circle</i>	Ordinarily you should not process double-circle; PenPoint uses it to create a reference button linked to the current selection.
^	<i>Double-Caret</i>	Don't process the double-caret; PenPoint uses it to bring up the Create menu.
	<i>Flick Up-Down</i>	Don't process up-down. Pass it on so that PenPoint can use it to toggle the borders of embedded applications.
=	<i>Flick Left-Right</i>	Toggle the mode (if your application has modes).
?	<i>Questionmark</i>	Display Quick Help for the object or region.

Guidelines for Core Gesture Variations



Multiple Taps

When appropriate, use multiple taps to select a hierarchy of objects, such as word/sentence/paragraph or word/paragraph/page in text.

Use double-tap for the object's second most important operation: for example, floating the page associated with a reference button, toggling a section open and closed, or selecting a larger unit.

Triple-tap is often used to restore an object to its default state. For example, triple-tapping on the title line of an option sheet restores the sheet to its normal size, triple-tapping on the label of a tab resynchronizes the label to the title of its associated document, and triple-tapping on a reference button re-links the button to the current selection. If appropriate, use triple-tap in this sense.



Multiple Flicks

Use to scroll successively farther. In most cases, only flick and double-flick are needed: Flick scrolls to the edge, double-flick scrolls all the way to the beginning or end. If your application deals with a more complex surface, you can use quadruple-flick to scroll to the extreme, leaving double-flick and triple-flick to scroll to some appropriate intermediate location.



Caret-Tap

Use for whatever the secondary meaning of insert or create may be in the context of your application.

In text, caret pops up an insertion pad, and caret-tap creates an embedded insertion pad.



Check-Tap

Use when appropriate for a secondary meaning related to options.



Circle-Tap

Use for whatever the secondary meaning of edit may be in the context of your application.

For example, if circle edits one cell of a table, circle-tap might edit the entire line.

In MiniNote, circle over untranslated ink translates and displays the translation in a pop-up edit pad, and circle-tap translates without displaying the pad for editing.



Circle-Line

Use to replace text by bringing up an empty edit pad for the object under the gesture.

Guidelines for Open Gestures



Arrows
(4 directions)

PenPoint uses the vertical arrows in text to increase or reduce the point size.
The vertical arrows are also appropriate for zooming and unzooming.
Horizontal arrows have no suggested interpretation.

Double-Arrows
(4 directions)

Use as variations or intensifiers of the single-shafted arrows.



Up-Right

In text, inserts a single character.



Up-Left

No suggested interpretation.



Down-Left

In text, inserts a paragraph break.



Right-Up

Use in text to capitalize the first letter.

Consider using in the border of tables and other surfaces to close white space.



Right-Down

Use in text to make the word lower case.

Consider using in the border of tables and other surfaces to open up white space.



Left-Up

Consider using in the border of tables and other surfaces to close white space.



Left-Down

Consider using in the border of tables and other surfaces to open up white space.



Down-Right-Flick

Use as an intensifier for down-right. In text, inserts a tab.



Down-Left-Flick

Use as an intensifier for down-left. In text, inserts a line break.



Right-Up-Flick

Use as an intensifier for right-up. In text, capitalizes the entire word.

Chapter 14 / Processing Handwriting

This chapter describes user interface issues related to the processing of handwritten characters by the PenPoint™ operating system's handwriting translation system.

Topics covered include constraining translation, and the relationship between the user interface and different translation objects.

► Using Handwriting Translators

The translation of handwritten input into ASCII codes is performed by software objects called handwriting translators.

Because the handwriting translator and the user interface that mediates between the user and the translator are separate, you can tailor each to suit the needs of your application.

However, while these two components are separable, they are also related, and need to be designed together. It is often helpful to the user to have conventions that associate looks with translators. This gives the user a visual cue as to what kind of behavior the field will exhibit.

The standard PenPoint fill-in fields and overwrite fields provide an example of a convention associating a particular kind of translator with a specific visual cue.

When building a translator, the client can either provide a grid that the translator will use to segment the input strokes into discrete characters, or tell the translator to use heuristics to perform the segmentation.

The two types of standard PenPoint text fields are designed around this distinction. Overwrite fields present visual segmentation cues to the user, and tell the translator to use that grid. Fill-in fields, on the other hand, present no visual segmentation cues to the user, and tell the translator to determine the segmentation via heuristics.

► Constraining Translation

This section discusses the role of constraints in designing handwriting translators.

✦ The Problem of Ambiguity

Even assuming that the translator can correctly segment the input strokes into characters, the problem of determining what the user intended in the face of multiple possible valid shapes remains. Examples of handwritten forms that are

indistinguishable in the absence of constraints include the circle (letter o or numeral zero?) and the vertical line (letter l or numeral one?).

The default behavior of the system is to “put like with like.” So, for example, a circle preceded by “347” would be translated as a zero, while a circle preceded by “leg” would be translated as the letter o.

Improving Translation with Context-Specific Constraints

The basic technique available to you to handle the ambiguity described above is to impose constraints on the interpretation of the input based on its context.

Think of a continuum of input contexts. At one end would be input areas that accept any character that the translation engine is capable of recognizing. An example of such an unconstrained context would be a word processor, in which numbers, lower and upper case letters, and punctuation are all valid input.

At the other end of the continuum would be fields that accept only a small, completely specified set of characters. An example of a highly constrained field would be a United States telephone number, which always takes the form ###-###-####. An even more highly constrained field might only accept a small set of values—for example, a list of states. (For lists shorter than about 20, consider using an explicit choice list instead of a handwriting translator.)

When building a translator, you can tailor it to the appropriate place on the spectrum. The irreducible design tradeoff is that the more you constrain the context, the more accurate the handwriting recognition will be, and, at the same time, the greater limitation you place on what the user can enter.

The PenPoint translation system permits a great deal of flexibility in constraining translators. The sections that follow describe the different types of constraints and different ways that you can apply those constraints.

Constraining to Letters, Numbers, Punctuation, or Symbols

You can specify that a given translator accept any combination of upper case alphabetic letters, lower case alphabetic letters, numbers, common punctuation, or symbols.

Constraining with Dictionary and Templates

The PenPoint operating system allows you to further constrain translators by means of several types of templates:

- ◆ **Dictionary.** The dictionary is actually drawn from two sources: the built-in PenPoint dictionary, and the personal dictionary that the user has specified as current in the **Dictionaries** page of the Settings notebook.
- ◆ **Character lists.** A list of ASCII characters.
- ◆ **Word lists.** A word list is effectively a small, special-purpose dictionary.

- ◆ **Pattern descriptions.** You can describe a pattern by means of a simple language. For example, suppose you define **P** as the set {A, B, C}. Then the pattern **###PPP** would accept any digit in the first three positions, and only the letters A-C in the second three positions.

Furthermore, you can specify the degree to which each template constrains the translator. There are three levels:

- ◆ **Ignore.** The template is ignored during translation.
- ◆ **Enable.** The template is used as one of the rules in the process of translation.
- ◆ **Veto.** The translator will return *only* words that match the template. If the input does not match the template closely enough, the unrecognized character symbol will be returned for each input character.

✦ **Guidelines for Using Constraints**

The basic guideline in designing translators is to constrain the input as much as you judge reasonable, given the expected use of the field.

When constraints reflect expected rather than required usage, it's a good idea to give the user some means of turning off the constraints temporarily to enter characters that don't fit the expected pattern. For example, in a word processor that usually makes use of the dictionary to aid translation, give the user some way to turn off the dictionary when entering a string that he or she knows is not in the dictionary, such as a license plate number.

Chapter 15 / Presenting Input Modes

This chapter describes the role of input modes in PenPoint applications, how to let the user control input modes, and how to present input mode feedback.

▼ The Role of Modes in PenPoint

✦ Spatial Modes

In the PenPoint™ operating system, different regions of the screen interpret input differently. This spatial modality is a characteristic shared by all graphical user interfaces, whether pen-based or mouse-based. Spatial modality is pervasive and usually transparent to the user: Menus pop up, resize corners drag, fields take ink, and so on.

The key to making spatial modes transparent is to always provide clear visual cues to indicate the different regions. The cue may be part of the background of the visual scene, as when a boundary outlines a large region, or it may be part of the foreground, as when a word or picture provides a target for the pen.

✦ Explicit Modes

Explicit modality means that the same region interprets input differently, depending on the setting of a user switch.

Explicit modes have been common in command-based interfaces in the form of the familiar edit and insert modes of keyboard-based editors. Explicit modes are also widely used in mouse-based GUIs. The classic example is the drawing application with modes for creating the different types of objects—circles, squares, text, and so on.

Modes have gotten a bad name in graphical user interfaces because they add to the conceptual complexity of the application, and open the door to a whole class of potentially serious errors when the user forgets what the current mode is.

At first blush, the pen seems to offer the prospect of eliminating explicit modes, because with the pen the user can more easily make different gestures that the program interprets as commands to create different types of objects. However, in practice, modes are not so easily dispensed with. One approach is to use a combination of spatial modality and other heuristics to interpret pen input. For example, is the stroke on white space or on another object? Is it a gesture? If not, is it a shape? If not, is it a character? If not, well, just dribble ink.

The problem is that the user must at all times bear in mind the rules that the program will apply in interpreting the pen-strokes. When the user's model and the application's model don't match, annoying mistakes occur: The user writes a word and the application responds with the feedback for "unrecognized gesture," the user makes a cross-out to delete an object and the application inserts ink, the user tries to make a circle and the application creates a scribble, and so on.

So explicit modes remain a valid user interface model in pen-centric programs. In fact, modes may well prove to be particularly useful in pen-based interfaces, because markup layers or annotation modes are useful on almost any type of document: forms, text, drawings, maps, and so on.

User Models

One of the most fundamental design decisions you need to make is what model your application will present to the user with respect to modes.

The following table distinguishes the most common models.

NOTE: This document uses the terms **ink mode** and **gesture mode** for the basic insert and edit modes. The use of these terms should *not* be taken as a recommendation for what terms you should use to describe your application's modes to the user.

Model	Usage and Examples
Primary Gesture Mode (with special areas for ink)	Pen strokes over the document are interpreted as gestures. Input is done in special writing or editing pads. Example: Simple text editor such as MiniText.
Primary Ink Mode (with Gesture Mode for browsing)	Even if the emphasis is on ink, there is almost always some editing to do, even if only selecting for move or copy. Editing aside, the core gesture mode allows easy browsing. Example: Simple note taker such as MiniNote.
Dual Modes: Gesture & Markup	Any application that supports a markup or annotation layer over a document. Examples: A fax viewer or a forms viewer with markup mode.
Dual Modes: Ink & Gesture	In this model, the ink and edit modes are presented as equal to the user. The usage pattern is that the user switches modes frequently. Example: Ink processor that includes a rich set of editing capabilities.
Multiple Modes	Examples: Drawing and painting applications that have modes for selecting, erasing, and various flavors of inserting. In addition to these familiar modes, you may want to organize your application around different translation modes . The most common are sketch (no translation), shapes (circles, squares, ellipses, and so on), and text. This list is not exhaustive. For example, you might have modes that recognize specialized shapes, such as musical notes in a composing program. Also, note that translation modes are not necessarily mutually exclusive.

Guideline Summary

The PenPoint operating system guidelines for using modes boil down to two rules. The first rule is:

- ◆ Always have a **core gesture mode** in which each core gesture behaves as expected. If the gesture's function is irrelevant at that screen location, the gesture should do nothing.

This core gesture mode may be the only mode your application has. If you do have more than one mode, then the second rule comes into play:

- ◆ Follow the dual command path principle for switching modes. First, present a mode switch to tell the user what the current mode is and allow access to all the other modes. Second, observe the toggle mode gesture (flick left-right) in the document's work area to toggle between core gesture mode and the next most important mode.

Because of the potential for confusion that accompanies any type of mode, and in view of the likelihood that pen-centric applications will increase their reliance on modes as markup layers become common, it is especially important to follow the guidelines outlined above.

The following sections expand on these guidelines.

Core Gesture Mode

The user may use core gesture mode as either an edit mode or a browse mode. The essential point is that the user must always have an easy, consistent way to put the document in a state in which he or she can rely on the familiar core gestures without any fear of surprise, disorientation, or loss of data. This allows the user to come to think of core gesture mode as the “safe” mode.

Mode Switch

It is essential to provide visual feedback so the user can tell at a glance what the current mode is. The most common way is to put a mode switch at the far right of the menu line:

Meeting Notes	<24>
Document Edit Options	

Meeting Notes	<24>
Document Edit Options	

Mode Switch at Right of
Menu Line

Later sections discuss variations in the location and presentation of the mode switch.

Mode Toggle Gesture

The PenPoint operating system provides a special gesture, the flick left-right gesture, to allow the user to quickly toggle between modes. If your application has more than two modes, use the flick left-right gesture to toggle between core gesture mode and the most useful of the other modes, or allow the user to set which modes to toggle between.

It is essential that all applications with input modes process the flick left-right gesture *in all modes*, for several reasons:

- ♦ **Dual Command Path.** The user needs to be able to get back to core gesture mode even when the borders are turned off. Remember that PenPoint is built around the concept of embedding, and that the user can turn the borders and controls off an embedded document at any time.
- ♦ **Efficiency.** The user needs a quick, easy way to toggle between ink and gesture modes. The gesture allows the user to do that easily, without needing to move the pen to a special region or to target carefully.
- ♦ **Consistency.** If each application accomplished this in a different way, the user would have to think about it each time. The user needs to be able to count on this gesture in order to be able to use it transparently.

For all of these reasons, the toggle mode gesture should be considered core from the standpoint of PenPoint's Embedded Document Architecture.

▼ Mode Switch Presentation

This section gives recommendations for presenting the mode switch.

⚡ Pictures versus Text

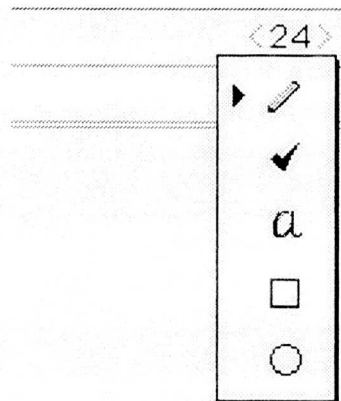
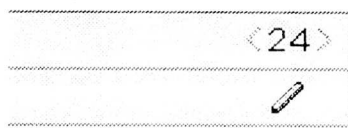
All of the examples given in this chapter use pictures rather than text. In a sense, this guideline is arbitrary because there is no inherent reason why pictures would be clearer than text.

We recommend pictures because they stand out, and are easily distinguished from the text labels of the menu line. They also take up less room, always an important consideration when designing applications for small, mobile computers.

⚡ Dual Modes: Toggle Switch

The PenPoint User Interface Toolkit provides a toggle control that shows a picture representing the current mode, and it switches when tapped. This is the recommended form for a two-state switch, because it is compact, and space on the menu line is at a premium.

The default pictures are a check for gesture mode and a pen for ink mode:



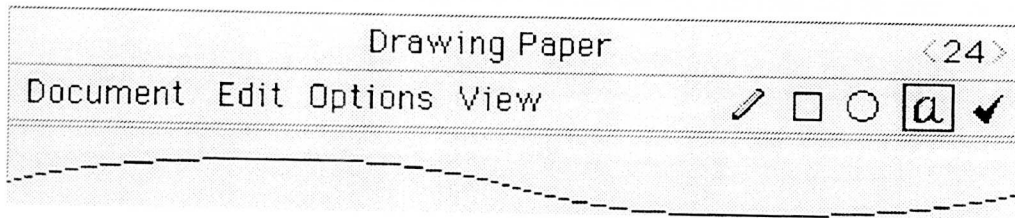
Default Mode Toggle
Switch

⚡ Multiple Exclusive Modes

For multiple modes that are mutually exclusive, use either a pop-up list or a boxed list.

Pop-Up List

The most compact way to show multiple modes is with a pop-up list:

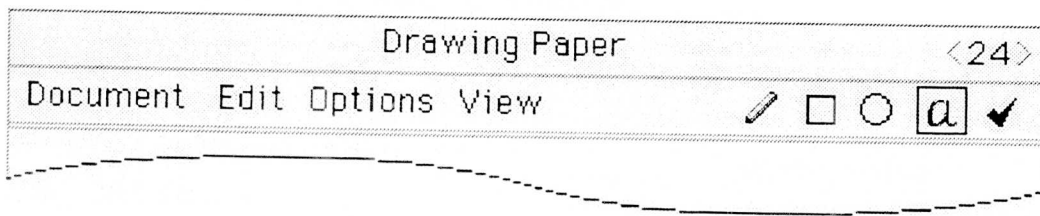


Pop-Up Mode List

The menu's label shows the current mode. The user can either tap to see the menu, or use flicks to switch modes without displaying the menu (flick up or left for next mode, flick down or right for previous, double-flick up or left for last, double-flick down or right for first).

Boxed List

For multiple modes, you can also use the standard PenPoint boxed list control:



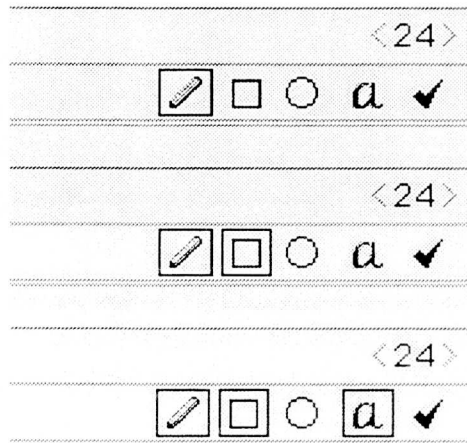
Boxed Mode List

A boxed list is more efficient to use than a pop-up list (because each mode is only a tap away) at the expense of taking up more room on the menu line.

⚡ Non-Exclusive Modes

If you have multiple modes that are additive rather than mutually exclusive, it is essential to show them all at once, so the user can see which are on at any given moment.

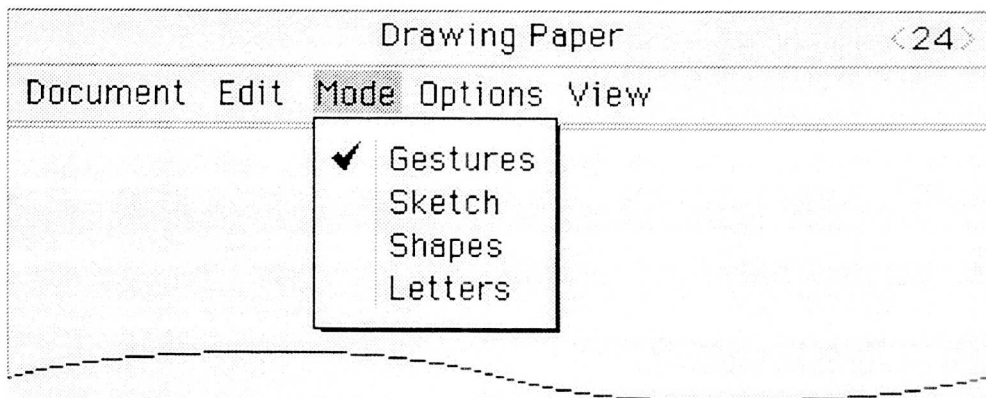
The example below shows translation modes that are non-exclusive:



Non-Exclusive Modes

⚡ Don't Hide the Mode Switch

Don't put the mode list in an ordinary menu:



X Not Recommended
Mode Switch in Menu

This forces the user to display the menu to discover what the current mode is.

Mode Switch Location

This section gives recommendations for where to put the mode switch.

Menu Line

The previous examples have all shown the default location at the right of the menu line. If your application has a subpage control in that location, then placing the mode switch right next to the subpage control may look a little cluttered:

Meeting Notes	<24>
Document Edit Options	 <1/3>

Mode Toggle Next to
SubPage Control

In that case, consider putting the mode switch immediately after the right-most menu label:

Meeting Notes	<24>
Document Edit Options 	<1/3>

Mode Toggle Following
Menus

If your application follows the “primary ink mode with gesture mode as backup” model, you may want to layer the functionality so that the novice user doesn’t have to learn about the backup mode right away. The way to do this is to add the mode switch to the list of controls that the user can show and hide on the Controls sheet:

Meeting Notes ▶ Controls

Show:

☒ Controls
☐ Borders Only

Controls:

☒ Menu Line
☒ Scroll Margin
☒ Cork Margin
☒ Mode Switch

Borders:

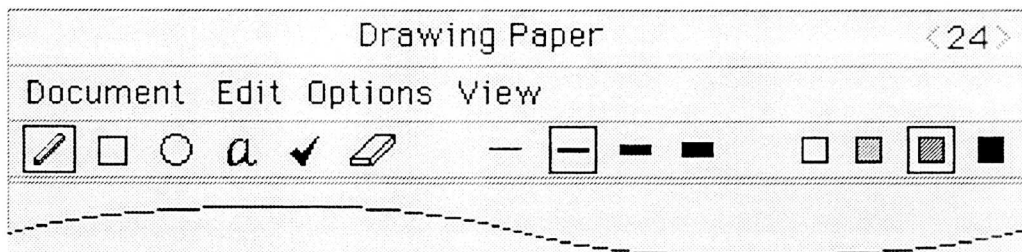
☒ Single

Apply
Apply & Close

Controls Option Sheet
with Option to Hide
Mode Switch

✦ Palette Line

If you have more modes than will fit at the right of the menu line, you can put the mode switch in a separate palette line:



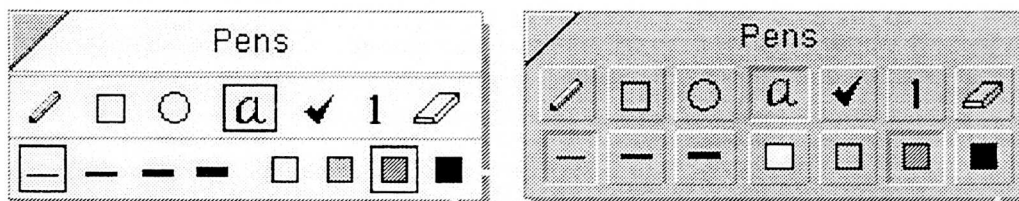
Mode Switch in
Palette Line

The palette line in the above example contains three groups of controls: input modes on the left, pen widths in the middle, and ink colors on the right.

Remember to add the palette line to the list of controls on the Controls sheet so the user can choose whether to display it or not.

✦ Pop-Up Palette

The figure below shows two versions of a pop-up palette with a mode switch and other controls:



Mode Switch in Pop-
Up Palettes (Two
Styles)

The above figure shows the standard paper-like visual style on the left, the grey "control-panel" look with raised buttons on the right. The control panel look is fine for such a small pop-up palette.

Examples of Modal Applications

This section outlines two examples of modal applications.

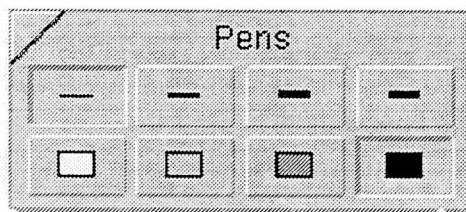
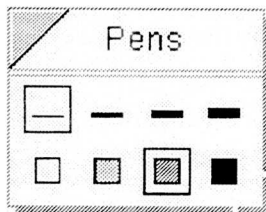
Markup Layer

The figure below shows the standard mode switch in the standard location. The check means gesture mode, the pen means markup mode:



Mode Switch for Markup Layer

When the user chooses markup mode, a palette of markup pens pops up:

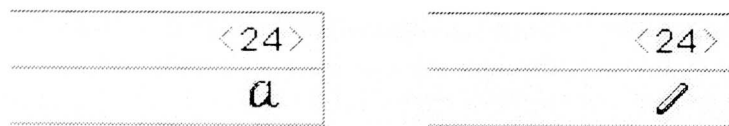


Pop-Up Palette of Markup Pens (Two Styles)

The pop-up palette has two benefits: It makes the markup controls available when applicable, and it also serves to provide additional feedback that the application is in markup mode.

Deferred and Immediate Translation

The figure below shows the mode switch for a forms package with deferred translation:



Mode Switch for Deferred Translation

The letter means that the program will translate pen input into text automatically; the pen means that what the user writes will be saved as ink, without being translated.

Such an application should allow the user to translate fields containing untranslated ink on a per-field basis.

Chapter 16 / Using Ink

The use of ink as a data type is one of the most promising areas for PenPoint applications. This chapter lists some of the ways ink can be used, and gives basic guidelines for editing ink.

▼ The Ink Data Object

The PenPoint™ operating system defines an object called a **scribble** for storing and scaling ink, and transferring ink between applications.

The PenPoint input system generates one or more scribbles each time the user writes on the screen. Depending on the context, the application may render the scribbles on the screen as ink, pass them to the translation system to be converted to text or gestures, store them for later translation, or ignore them completely.

▼ Ways of Using Ink

People use ink in countless ways. Ink can be casual and temporary, as in doodles and lists. It can serve as a way of identifying someone, as in a signature. Ink is a familiar and convenient way to mark up a document being worked on. In some contexts, ink remains only until it is translated into printed text, as in a field in a form. In a drawing or sketch, on the other hand, ink may be the final means of expression. With ink, people routinely mix text, calculations, pictures, and annotations on a single sheet.

The table below distinguishes four levels of ink functionality, ranging from plain ink that cannot be edited in any way, through ink that is fully editable and translatable.

Ways of Using Ink

Category	Purpose and Functionality	Examples
Plain	For casual and ephemeral uses. Cannot be edited or translated in any way.	Doodles, signatures, simple annotations.
Translatable	Transitory state: The ink is to be translated, but not edited in the meantime.	Deferred translation on a form.
Editable	The user can edit the ink by applying the basic editing operations (Move, Copy, and Delete) as well as by modifying the ink's appearance (making it bold, changing the color).	Markup layer, simple notetaker.
Editable and Translatable	Ink may be edited or translated.	Full-function ink processor.

The table gives a framework for thinking about how to use ink in your application. In particular, consider whether these three ways of using ink may be relevant to your application:

- ♦ **Markup layers.** Almost any type of document—letter, memo, report, spreadsheet, engineering drawing, map, form, contract, and so on—can benefit by the addition of a markup or annotation capability.
Markup functionality is typically implemented as a separate input mode, as described in Chapter 15.
- ♦ **Deferred translation.** Being forced to translate at the time of writing may be a burden, particularly for the mobile user who is interacting with others. Taking notes in a meeting or filling out a form during an interview are classic examples. In such situations, the user needs to be able to write without translating, and defer the translation process until it is convenient.
- ♦ **Editing ink.** PenPoint includes an ink component, called Notepaper, and a simple application built around it, called MiniNote. The features of MiniNote are described in Chapter 18 of Book I.

MiniNote just scratches the surface of what promises to become a new class of ink processing applications that allow the user to manipulate untranslated ink as easily as translated text.

■ Guidelines for Editing Ink

The basic challenge in designing ink editors is to combine the simplicity of pen and paper with the power and convenience of the computer.

In addition to the primary mode, in which pen input is simply stored as ink, ink editors should implement a gesture mode, as described in Chapter 15. These two modes are described below.

■ Ink Mode

The strength of ink mode is its simplicity and familiarity: The user simply writes or draws, as with a pen and paper.

As compelling as the use of the pen to enter data is, remember that the pen is also a device for entering commands. The key to smoothly integrating these two basic functions is to allow selection in ink mode, as described below.

■ Gestures to Select

Allow the user to select scribble objects in ink mode with the following two gestures:

- ♦ Double-tap to select a single object.
- ♦ Tap-press to begin an area select.

While it is essential to provide an easy way to select in ink mode, it is also essential to avoid inadvertent selection. That is why tap should not be used for selection if the period character or a single dot is legal input. The only way to distinguish between a period and a tap to select is by timing, which is not usable in practice. If the user writes, then pauses too long before writing a period, the period will be taken as a tap gesture; if the user taps too quickly after writing, the tap will be taken as a period.

The press gesture for area select is prone to a similar problem. If the user touches pen to screen and then pauses too long before writing, an unintended area select will begin; if the user wants to select an area but doesn't pause quite long enough, an unwanted scribble will result.

Gestures over the Selection

Allow the user to make gestures over selected objects in ink mode, including at least the 11 core gestures, and any other gestures as appropriate.

Allowing the user to select, and to gesture over the selection, makes available the functionality of gestures without requiring the relatively heavyweight operation of a mode change.

Flick Left-Right Gesture to Toggle Modes

Accept the flick left-right gesture anywhere to toggle between ink and gesture modes. Remember that the user may embed your application in another document and turn off the title line and menu line. In this situation, the only way for the user to turn the borders back on is to use the toggle mode gesture to get back to gesture mode.

Other Gestures

Depending on the context of your application, it may be appropriate to provide a few other gestures in ink mode. The PenPoint ink component, for example, accepts the scratch out gesture in ink mode to delete unselected objects.

Take care that any such additional gestures do not interfere with writing or compromise the simplicity of the pen and paper model.

Pen Styles

Allow the user to choose pen styles as appropriate to your application, including different colors, thicknesses, and nib styles (italic, for example).

✦ Gesture Mode

In gesture mode, the user must at least be able to make the 11 core PenPoint gestures.

Your application should also support the non-core gestures that have well-defined meanings in PenPoint, such as B for bold and N for normal style, double-caret to create an embedded document, double-circle to create a reference button, flick up-down to toggle borders on and off, plus to toggle an object's selected state, and brackets to adjust an existing selection.

Use the gestures in the circle family as follows:

- ◆ Circle should translate the selected ink and pop up an edit pad containing the results of the translation. If the selection is already translated, then the label on the edit menu should read **Edit**. If the current selection is *not* translated, the menu label should read **Translate & Edit**.
- ◆ Circle-tap should translate the selected ink and insert the results directly into the document, *without* bringing up an edit pad.
- ◆ Circle-line should pop up an edit pad to replace the selected ink or text. If the selection is translated, use the standard PenPoint edit pad. If the selection is *not* translated, use a pad that replaces the selection with untranslated ink.

Use other gestures as appropriate to manipulate both ink and text. For example, MiniNote has gestures for aligning lines, for evening out the spaces between objects on a line, for grouping and ungrouping, and for opening and closing white space.

✦ Gesture Margin

The PenPoint ink component provides a **gesture margin** to make it convenient to select and operate on entire lines.

If you implement such a gesture margin, remember to allow the user to show and hide it from the **Controls** option sheet.

Summary of Ink Gestures

The table that follows gives guidelines for using gestures in ink editors.

Ink Editing Gestures

Gestures for Ink Mode



Double-Tap. Select a single object.



Tap-Press. Initiate an area select.



Flick Left-Right. Toggle between ink and gesture modes.

Core Gestures for Gesture Mode (or over selection in Ink Mode)



Tap. Select a single object.



Press. Initiate drag move when made over selected object. Initiate area select when made over background or unselected objects.



Tap-Press. Initiate drag copy.



Flick. (Four directions.) Scroll to edge.



Cross Out Delete a single object.



Caret. Insert or create the most natural object in the context.



Circle. Edit (when made over translated text); **Edit & Translate** (when made over untranslated ink).



Check. Display the object's option sheet.



Undo. Equivalent to choosing **Undo** from the **Edit** menu.



Pigtail. Delete a single character in text, delete smallest object of a compound object.



Down-Right. When made over translated text, insert a single space. When made over untranslated scribbles, insert space depending on the length of the horizontal leg of the gesture.

Ink Editing Gestures (Continued)

Non-Core Gestures for Gesture Mode (or over selection in Ink Mode)



Double-Tap. Select a single object.



Triple-Tap. Select a line (or larger unit).



Quadruple-Tap. Select a paragraph (or larger unit).



Plus. Toggle selected state of object.



Brackets. Adjust existing selection.



Circle-Tap. Translate ink to text without displaying an edit pad.



Circle-Line. Brings up an empty editing pad for word or scribble.



Double-Circle. Create reference button linked to the current selection.



Right-Down and Left-Down. Open white space as determined by the length of the vertical leg.



Right-Up and Left-Up. Close space as determined by the length of the vertical leg.



Double-Flick. (Four directions.) Scroll to beginning/end.



Flick Up-Down. Toggle borders of embedded document on and off.



Make object (or selection) bold.



Makes the word (or selection) “normal”—that is, turns off the bold and any other attribute.

Part 5

Viewing and

Manipulating

Application Objects

▼ Chapter 17 / Scrolling and Zooming	263	▼ Chapter 21 / Moving and Copying	285
Scrolling	263	User Model: Drag & Drop	285
Scroll Margin Operations	263	Move and Copy in Mouse-Based Interfaces	285
Showing and Hiding Scroll Margins	264	Move and Copy in PenPoint	286
Scrolling by Flicking	264	Initiating the Drag	286
Zooming	265	The Drag Icon	286
Zooming Operations	265	Dragging Large Objects	287
Gestures for Zooming	266	Dragging Offscreen Objects	287
Zoom Control in Menu Line	266	Completing the Drag	288
Floating Navigation Control Area	267	Drag Rectangle	288
▼ Chapter 18 / Scaling and Resizing	268	Targeting the Destination	288
Scaling Strategies	268	Moving and Copying Between Applications	289
Scaling with the System Font	269	Standard Data Types	289
Using Layout Units	269	Refusing a Data Transfer	289
Handling PenPoint Building Blocks that Don't Scale	269	Variations on Move and Copy	289
Allowing the User to Specify the Font	271	Move as Adjust	289
Resizing	272	Mapping Move to Copy	290
Resize Handles	272	Examples	291
Minimum and Maximum Sizes	272	▼ Appendix A / PenPoint Design Checklist	295
Scaling on Resize	273		
▼ Chapter 19 / Optimizing Layout for Small Screens	274		
Minimum Screen Size	274		
Minimizing Horizontal Scrolling	274		
Orientation-Specific Layouts	276		
▼ Chapter 20 / Selecting	278		
User Model	278		
Selection Feedback	278		
Don't Use Inversion	278		
Feedback for Different Types of Objects	278		
Selecting and Deselecting	279		
Selecting a Single Object	279		
Deselecting an Existing Selection	280		
Extending a Selection by Dragging	280		
Extending a Selection with Brackets	280		
Discontiguous Multiple Selections	282		
Auto-Selection of Gesture Targets	283		
Text Selection	283		
Selection in Option Sheets	283		

Chapter 17 / Scrolling and Zooming

Often the information the user wants to see will not fit in the available viewing region. This is a common situation for desktop computers, and will be even more common in mobile computers with pocket-sized displays.

The two primary techniques for dealing with this are scrolling and zooming. In scrolling, the user moves the information past the viewing region, either horizontally or vertically. Zooming involves the third dimension of the perceived distance to the object being viewed. By analogy to a telephoto lens, the user can “zoom in” closer to the object.

This chapter gives very specific guidelines for scrolling, and more general guidelines and suggestions for zooming.

■ Scrolling

As with all important operations in the PenPoint™ operating system, there is a dual path for issuing scrolling commands. The scroll margin with its controls is complemented by the flick gestures.

■ Scroll Margin Operations

The user can choose between two different styles of scroll margins from the **Fonts & Layout** page of the Settings notebook.

The **Arrows & Drag Box** style supports the following operations:

- ◆ **Scrolling by unit.** Interpret a tap on one of the scrolling arrows as a request by the user to scroll by one unit. In applications that are organized by lines, such as text or lists, the line is the obvious unit. For tables, the unit should be a row or column. For a painting application, the unit might be as small as a single pixel.
- ◆ **Scroll by screenful.** A single tap above the drag box should scroll towards the beginning of the sheet by one screenful. A tap below the drag box should scroll towards the end of the sheet by one screenful.
- ◆ **Thumb.** When the user drags the drag box, scroll to the corresponding location in the sheet. Dragging the box to the top should result in a scroll to the beginning of the sheet, dragging half way should result in a scroll to the half-way point of the sheet, and so on.

- ◆ **Scroll to beginning/end.** A double-tap above the drag box should scroll all the way to the beginning of the sheet. A double-tap below the drag box should scroll all the way to the end of the sheet.

The **Tap & Flick** style of scroll margin has no visible controls. It is simply an area for scrolling gestures. The operations are:

- ◆ **Line to top/bottom.** Flick up should scroll the line under the beginning of the flick to the top viewing region. Flick down should scroll the line to the bottom.
- ◆ **Scroll to beginning/end.** Double-flick up should scroll to the beginning of the sheet, double-flick down should scroll to the end.
- ◆ **Thumb.** Tap is equivalent to dragging the drag box in the other style of scroll margin.
- ◆ **Scroll to beginning/end.** A double-tap in the top half of the scroll margin should scroll all the way to the beginning of the sheet. A double-tap in the bottom half should scroll all the way to the end of the sheet.

✦ Showing and Hiding Scroll Margins

It is best to display scroll margins only as needed. If all of the data is showing, don't show a scroll margin.

✦ Scrolling by Flicking

The user can scroll by making flick gestures over the **Tap & Flick** style of scroll margin, as described in the previous section.

As described in the Chapter 16, you should always provide a mode in which the user can make flicks (as well as the other core gestures) over the document itself.

The model presented to the user is that he or she is manipulating a sheet of paper directly by shoving it with the pen. Therefore, flicking *up* should always move the contents of the sheet *up*, and reveal more information that had been hidden past the *bottom* edge. Likewise, flicking *down* should always move the contents of the sheet *down*, and reveal more information that had been hidden past the *top* edge.

The two most important scrolling gestures are the single and double flicks. In most cases, you should interpret them as follows:

- ◆ **Single Flick.** Send the line or point under the start of the flick to the edge of the viewing region. So flicking up on a line should send it to the top edge, and flicking down on line should send it to the bottom edge.
- ◆ **Double Flick.** Scroll as far as possible. That is, double-flicking up should scroll to the end of the sheet, and double-flicking down should scroll to the beginning of the sheet.

The single and double flicks should suffice for most applications. However, PenPoint also supports the triple and quadruple flick gestures. If your application is organized hierarchically, with more levels of structure, you may want to use the double and triple flick to scroll to intermediate levels, and quadruple flick to scroll all the way to the beginning or end.

■ **Zooming**

Zooming is an alternative to scrolling for dealing with the problem of viewing a large surface through a small window.

As smaller devices come into wide use, zooming may prove to be just as important as scrolling, or even more so. Think of a person in the field who needs to view a large data surface, such as a map, blueprint, or spreadsheet, on a pocket-sized screen.

While PenPoint 1.0 does not define a standard interface for zooming, this chapter describes the most common zooming operations and gives suggestions for implementing them.

There are many approaches you can take. The basic point is to make zooming easy and quick. Whatever your design, provide a dual command path for zooming that offers both gestures and visible controls such as menus or buttons.

■ **Zooming Operations**

Useful zooming operations include:

- ◆ **Zoom in to region.** The user specifies a region by dragging out a bounding box, and the system zooms in so that region fills the display region.
- ◆ **Zoom in to point.** The user indicates a point, and the system zooms in by a fixed percentage to that point.
- ◆ **Zoom in or out by percentage.** The user either indicates the desired absolute percentage, or zooms in or out by specified steps. The same point stays in the center of the display region.
- ◆ **Actual size.** Show the document in its actual physical size.
- ◆ **Fit height and width.** Fit width is particularly useful for documents that read vertically, as it uses all the available horizontal screen space without requiring horizontal scrolling.
- ◆ **Show all.** Zoom out to show the entire document.

Gestures for Zooming

The following table gives suggestions for gestures to use for zooming.

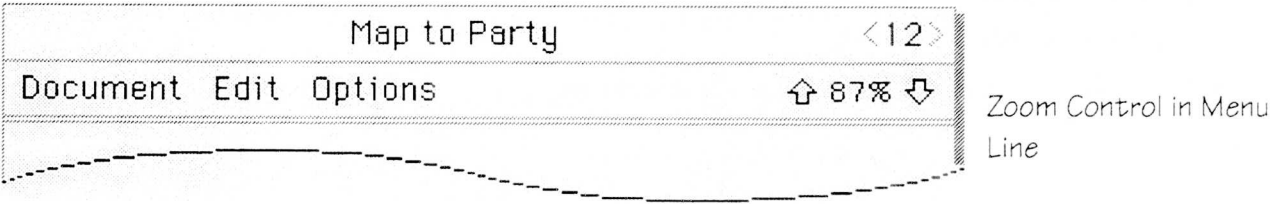
Gestures for Zooming		
	Press	Trace out a rectangle and zoom to it as soon as the user lifts the pen. NOTE: Press is the standard gesture for area selection. If your application supports area selection, use press for that purpose.
	Tap-Press	Trace out a rectangle that remains on the screen after the user lifts the pen. The rectangle should be draggable and have resize corners so the user can adjust it. After adjusting the rectangle as desired, the user taps inside the rectangle to zoom in, outside to dismiss the rectangle.
	Triple-Tap	Zoom to actual size. This is in keeping with the use of triple-tap elsewhere in PenPoint to mean "restore the object to its default state."
	Arrow Up	Zoom in by a specified percentage around either the tip of the arrow or the center of the viewing window.
	Arrow Down	Zoom out a specified percentage.
	Double-Arrow Up	Zoom in as far as allowed.
	Double-Arrow Down	Zoom out all the way (show the entire document).

Tap and double-tap are not included in the above table because they are typically used for other purposes, but it may make sense to use them for zooming. With the tap gestures, it's a matter of which are the most important operations to support for your particular application.

For example, take the case of a read-only map viewer. The only reason to support selection would be so that the user can select a point in the map and make a reference button to it. In that case, it would be appropriate to use tap and double-tap for zooming operations, and fall back on the other standard selection gesture, the plus, for selection.

Zoom Control in Menu Line

The figure below shows a zoom control at the right of the menu line:



Tapping the up arrow zooms in, tapping the down arrow zooms out. Pressing the arrows steps through the percentages without redisplaying, so the user can skip steps.

Tapping on the percentage displays a menu with other zoom commands:



Zoom Control with Menu

The **Zoom to Region** command puts up a draggable and resizable zoom box as described under the tap-press gesture in the previous section.

Floating Navigation Control Area

Another fruitful direction is to have a floating control for zooming in and out. Such a control would represent the document in miniature and allow gestures to be made on the representation, and might also have additional buttons or menus.

Chapter 18 / Scaling and Resizing

The PenPoint™ operating system is very flexible with respect to the parameters that affect the presentation of information. Factors that can change include:

- ◆ PenPoint machines vary in both the size and the resolution of the display.
- ◆ The user can change both the size and face of the system font at any time from the Settings notebook.
- ◆ The user can change the orientation of the display to portrait or landscape at any time from the Settings notebook.
- ◆ The user can change the size of the document at any time by resizing the notebook (or resizing the document itself, if it is floating).

This flexibility poses special problems with respect to layout of the application's visual elements. The bottom line is that you do not know and cannot control how much display space will be available to your application at any given time.

This chapter covers a range of layout-related issues, including scaling with the system font, allowing the user to specify the font, and resizing.

▀ Scaling Strategies

All the standard user interface components scale with the system font. This includes everything that uses the system font, such as document titles, menus, option sheets, error notes, notebook tabs, and reference buttons. It also affects the size of non-textual visual elements such as the amount of white space between visual elements, the thickness of borders and dividing lines, and the width of scroll margins.

When it comes to the body of your document or accessory, there are two basic approaches you can take with respect to the system font size:

- ◆ Scale all the visual elements relative to the system font.
- ◆ Scale all the visual elements relative to an absolute size related to the actual size of the document in its printed form. Typically, this means allowing the user to set the point size of text, or providing a zoom feature that allows the user to move the viewing perspective closer or farther from the data surface.

▀ **Scaling with the System Font**

This section describes how to respond to changes in the system font.

▀ **Using Layout Units**

If you are following the system font, the most important rule is to specify the locations of all objects in your application in terms of **layout units**, rather than absolute units of measure (such as inches, millimeters, or points) or units for a particular device (such as pixels).

Layout units are device-independent, and are driven by the system font.

▀ **Handling PenPoint Building Blocks that Don't Scale**

There are several user interface components that don't automatically scale in the 1.0 release of PenPoint. They include fill-in fields, overwrite fields, the text component, the ink component, and signature pads. Most of these components will be enhanced to observe the system font size in future releases of PenPoint.

In the meantime, you need to scale them yourself if you use them together with elements that *do* follow the system font and you want everything to scale uniformly.

For example, suppose you are using the text component to present a scrollable text region, along with some labels and checkboxes to implement a form. The figures on the following page illustrate what happens when only some of the visual elements scale.

Everything looks fine at the default 12-point font:

Service:

☐ Unsurpassable	☐ Extremely Competent
☐ Superb	☐ Very Good
☐ Excellent	☐ More Than Adequate

Comments:

Blah, Blah blah, had a good time, looking forward to next visit, blah blah blah, special thanks to Maurice for the shoes.

Thank you for using Amalgamated!

Form Looks Fine at 12 Points

When the user chooses the 18-point font, the visual harmony is spoiled by the non-scaling text box.

Service:

☐ Unsurpassable	☐ Extremely Competent
☐ Superb	☐ Very Good
☐ Excellent	☐ More Than Adequate

Comments:

Blah, Blah blah, had a good time, looking forward to next visit, blah blah blah, special thanks to Maurice

Thank you for using Amalgamated!

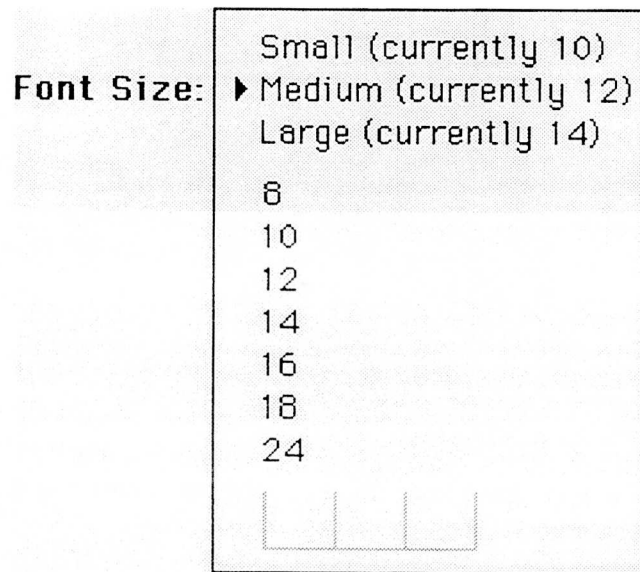
Scaling Problems Appear at 18 Points

To make the text box scale properly, your application would need to set the text to the system font size, and also set the dimensions of the box appropriately.

▀ Allowing the User to Specify the Font

If you allow the user to specify the font size for your application, it is a good idea to permit the specification either as an absolute size in points or relative to the current system font size.

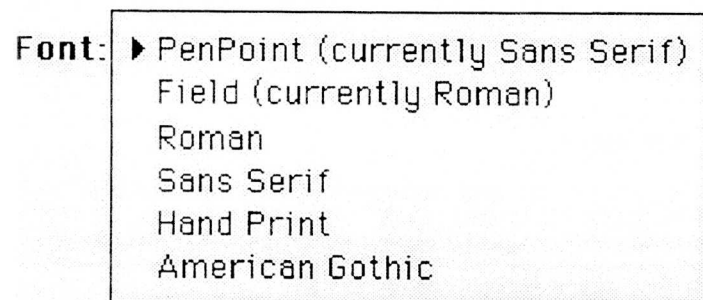
Here's how a menu to specify font size might look:



Font Size Menu

The first three entries are relative to the current system font size, which in the example is 12 points. When the user changes the size of the system font, the first three entries should change accordingly.

The same approach of presenting both absolute and relative choices is recommended if you allow the user to set the font face:



Font Face Menu

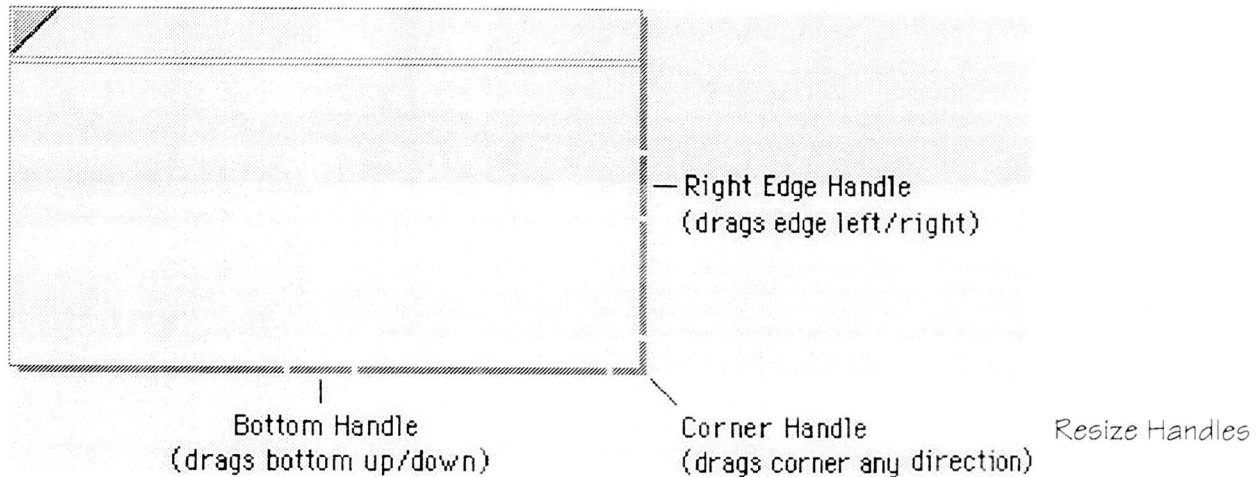
The first two entries represent the current settings of the **PenPoint Font** and **Field Font** preferences, respectively, on the **Fonts & Layout** page of the Settings notebook.

➤ Resizing

This section describes how to respond when the user resizes the viewing region.

⚡ Resize Handles

You can specify that pop-up sheets or accessories have resize handles. There are three handles, as shown in the following figure:



All the handles are optional. The choice of which handles to provide depends on the data. Here are some guidelines:

- ◆ Use a corner handle when any aspect ratio makes sense. Examples: pop-up writing and edit pads.
- ◆ Use a bottom handle only when the width is fixed. Examples: option sheets, embedded writing pads, and embedded text boxes.
- ◆ Use both a bottom and corner handle when arbitrary resizing makes sense, but the constrained resizing is an added convenience. Example: floating documents.

⚡ Minimum and Maximum Sizes

You should always specify a minimum and maximum size for a resizable object.

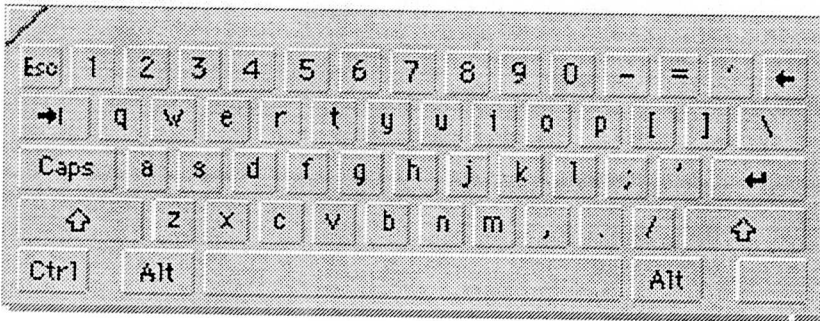
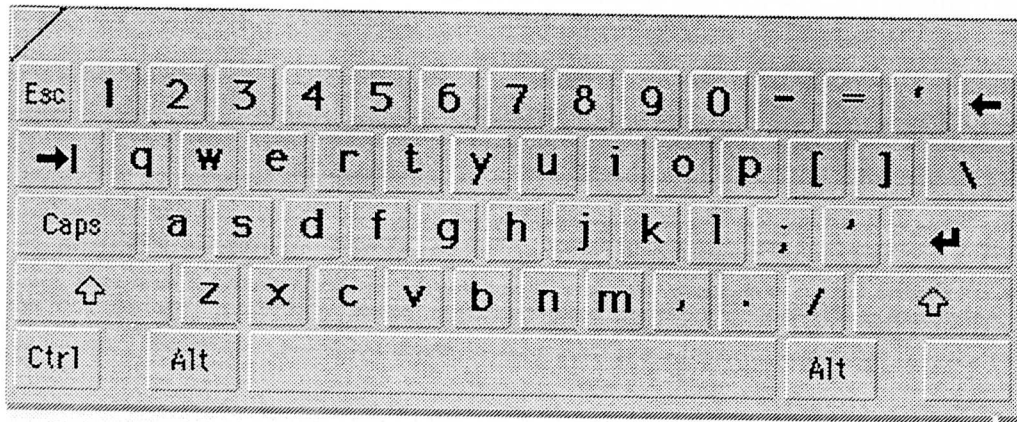
Make the minimum size the smallest size in which the object is still usable. For a text sheet, the minimum size might be tall enough to show the title and a single line of text, and wide enough to show a few words. For an option sheet, the minimum height should allow enough room to show the title line, one control, and the command line.

It is often important to specify a maximum size as well. For example, if a pop-up edit pad contains a field that is limited to 32 characters, the user should not be able to make it bigger than that.

⚡ Scaling on Resize

The simplest thing to do when the user resizes a sheet or accessory is to show more or less of the data, adding scroll margins if it is not all visible.

Occasionally it makes more sense to show the same data, and scale it to fit the size specified by the user, as in this keyboard accessory that scales when the user resizes it:



Keyboard that Scales
on Resize

Chapter 19 / Optimizing Layout for Small Screens

This chapter defines the minimum screen size and resolution for PenPoint computers, and gives some guidelines for optimizing layout when the screen area is limited.

Minimum Screen Size

While scalability is a goal of the PenPoint™ operating system, and should be a goal of all PenPoint applications, there is a practical limit to how small things can get. PenPoint defines that limit as follows:

- ◆ Number of pixels. The smallest supported screen size is 320 x 200 pixels.
- ◆ Size of pixels. Pixel size is commonly referred to as **resolution**, and measured in **dots per inch (dpi)**. The minimum display resolution is 72 dpi.
- ◆ Size of system font. This parameter is related to the resolution. At 72 dpi, the smallest font that is clearly readable to most people is 10 points.

These three parameters—320 x 200 pixels, 72 dpi, 10-point font—together determine the lower bound for display parameters. Your application needs to be prepared to display itself well under these constraints.

There's no need to define a limit at the other end of the size scale. If the display has more pixels, either the same size or smaller, then more information will be readable.

Minimizing Horizontal Scrolling

Horizontal scrolling is generally less convenient for the user than vertical scrolling. This is particularly true for text or tables, which (in most languages) are scanned horizontally, line-by-line. If the entire line of text or row of a table is not visible, there's no way to scan the table without scrolling back and forth repeatedly.

You can often minimize the need for horizontal scrolling by keeping the most important elements close together, rather than widely separated horizontally.

To illustrate, let's take a table similar to the Installed Handwriting page of the Settings notebook. The two required columns are **Name**, showing the the name of the handwriting profile, and **Current**, which is a column of checkboxes allowing the user to specify the profile to be used by the system to translate handwriting. There are several other columns that the user can show or hide.

Putting the two most important columns, Name and Current, at the far left and far right ensures that if the user makes the sheet narrower it will be more clumsy to use. As shown at the right in the figure below, when the sheet is narrow, the Current column is scrolled out of view:

Handwriting				
Name	Type	Size	Modified	Current
Standard	GOWrite	1596	No	☐
Arnold	GOWrite	1005	No	☐
Zoot	GOWrite	1440	No	☒
TrueWrite	TrueWrite	1668	No	☐

Handwriting	
Name	Type
Standard	GOWrite
Arnold	GOWrite
Zoot	GOWrite
TrueWrite	TrueWrite

X Not Recommended
Separating Primary
Columns Leads to
Unnecessary
Horizontal Scrolling

Putting the two primary columns next to each other allows the user to shrink the sheet and still use it conveniently:

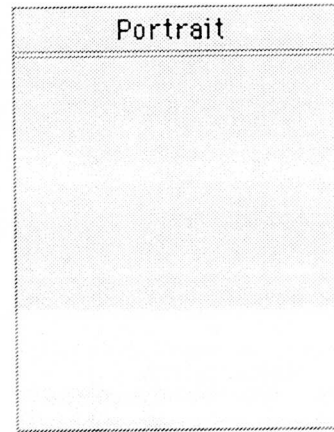
Handwriting				
Current	Name	Type	Size	Modified
☐	Standard	GOWrite	1596	No
☐	Arnold	GOWrite	1005	No
☒	Zoot	GOWrite	1440	No
☐	TrueWrite	TrueWrite	1668	No

Handwriting	
Current	Name
☐	Standard
☐	Arnold
☒	Zoot
☐	TrueWrite

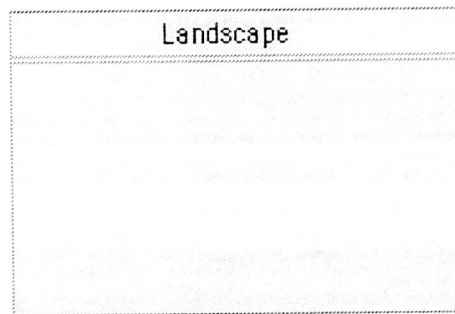
Put Primary Columns
Next to Each Other to
Minimize Horizontal
Scrolling

Orientation-Specific Layouts

A viewing area that is taller than it is wide is said to be in **portrait orientation**; a viewing area that is wider than it is tall is in **landscape orientation**:



Portrait Orientation



Landscape Orientation

Portrait and landscape aspect ratios can result from either of two types of user action:

- ◆ The user can specify that the display is oriented to any of the four edges of the screen via the **Top Edge** control on the **Fonts & Layout** page of the Settings notebook. Assuming square pixels and rectangular displays, the four possible settings resolve to the two portrait and landscape orientations.
- ◆ The user can resize a floating document or accessory.

In many cases, the orientation will not affect the layout. For example, text typically wraps at the right edge and scrolls vertically regardless of the orientation.

However, in some cases it may be possible to optimize your application's layout based on the current orientation, as in the example below:

Calendar < 4 >

July, 1994

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Wednesday, July 3

8:00
9:00
10:00
11:00
12:00
1:00
2:00

Layout Optimized for
Portrait Orientation

Calendar < 4 >

July, 1994

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Wednesday, July 3

8:00
9:00
10:00
11:00
12:00
1:00
2:00
3:00
4:00

Layout Optimized for
Landscape Orientation

Chapter 20 / Selecting

This chapter gives guidelines for managing the system-wide **current selection**.

Topics covered include:

- ◆ **User Model.** Definition of selection from the user's viewpoint.
- ◆ **Selection Feedback.** Standard selection feedback for different types of objects.
- ◆ **Selecting and Deselecting.** Topics include selecting a single object, adjusting an existing selection, selecting groups of objects, selecting multiple discontinuous objects, and auto-selection of gesture targets.
- ◆ **Text Selection.** Input from the keyboard should delete any existing text selection.
- ◆ **Option Sheet Selection.** Handling option sheets that themselves contain objects that take the selection.

▼ **User Model**

Graphical user interfaces generally present the user with what has been called the “noun-verb” model for invoking operations. In this model, the user first identifies the object of interest (the noun), then indicates which operation to apply to that object (the verb).

The act of identifying the operand is called “selecting an object,” or “making a selection.” The object, once selected, is called the **current selection**.

▼ **Selection Feedback**

The current selection must be distinguished visually from the other objects on the screen so that the user can identify it at a glance.

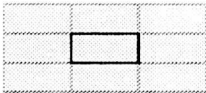
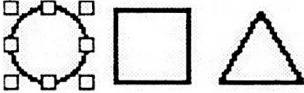
⚡ **Don't Use Inversion**

It may be easy to simply invert the selected object's rectangle to show selection. Resist this temptation, because gestures that the user makes will not be visible over the black mass of the selection.

⚡ **Feedback for Different Types of Objects**

While it is desirable for selection feedback to be as consistent as possible, no one convention suffices for all types of objects. The following table shows selection feedback for several types of objects.

Selection Feedback for Different Object Types

Object	Selection Feedback	Example
Text	Light grey background.	Four score and seven years ago.
Item in a List	Light grey background. Usually it looks best for the background to extend the full width of the list.	A Clockwork Orange Planet of the Apes China Blue
Cell in a Table	Outline the selected cell or block of cells in a darker (and thicker, if necessary) outline. This does not interfere with the display of the contents of the cell.	
Icon	PenPoint underlines selected icons.	? ✓ ↩ ↪ 📄 🕒 ⚙️ ⬇️ ⬆️
Resizable Figure	Indicate the selected figure by the presence of resize handles at the corners and at the midpoint of the edges of the figure's bounding rectangle.	

Selecting and Deselecting

This section describes the PenPoint conventions for selecting and deselecting objects.

The importance of selection is reflected in the fact that all mouse-based interfaces use the main mouse button to select objects. Those interfaces that came out of the Xerox tradition go further, and devote two mouse buttons to selecting: one to making new selections and the other to adjusting existing selections.

One of the strengths of the pen versus the mouse is that it permits the user to easily make gestures other than tapping. The PenPoint™ operating system takes advantage of this strength by providing a variety of complementary ways to make and adjust selections. Selection-related gestures include tap, double-tap, plus, press-drag, and left and right brackets.

Selecting a Single Object

The basic gesture for selecting a single object is the tap. Tapping a figure in a drawing should select the figure, tapping an item in a list should select the item, tapping a cell in a table should select the cell.

Some types of objects function like controls in that they use the tap gesture to invoke their main operation. So, for example, tapping on an icon opens the icon, and tapping on a tab or a reference button turns to the associated document.

In these “control-like” cases, where selection is not as important as the object’s primary function, you should interpret plus as the selection gesture.

✦ Deselecting an Existing Selection

From the user's perspective, the current selection is only useful in relation to an operation the user intends to perform in the very near future. At other times, it may appear as an artifact that is irrelevant, or even distracting.

Therefore it is important that the user be able to easily deselect any object.

Always allow the user to deselect in two ways:

- ◆ **Tapping on the object.** Tapping (or making the plus gesture) on an existing selection should deselect it.
- ◆ **Tapping on the background.** Tapping anywhere on the background of the region should also deselect any existing selection. Examples include the background of a drawing region, or the dotted lines connecting document names with their page numbers in the notebook table of contents, or the white space past the bottom of a list.

✦ Extending a Selection by Dragging

In many contexts the user can select multiple objects: a span of text, a block of items in a list, a block of cells in a spreadsheet, several figures in a drawing, and so on.

If your application supports multiple selection, you should allow the user to select adjacent objects using the press-hold gesture. The user touches the screen, waits for the press-hold timeout to elapse, and then drags out the region to select with the pen.

Use one of these two conventions for providing feedback during the drag:

- ◆ **Wipe-through.** If your application uses a light grey background to indicate selection, the grey should follow the pen as the user drags, jumping from object to object. This is often called "making a wipe-through selection." Examples include jumping from letter to letter in text and from item to item in a list.
- ◆ **Bounding Box.** If your application uses some form of outline or underline to indicate selection, use a bounding box to provide feedback as the user drags. Examples include selecting a group of adjacent figures in a drawing, a block of pixels in a pixel-based paint program, or a block of cells in a table.

✦ Extending a Selection with Brackets

Use the left and right bracket gestures to allow the user to adjust an existing selection in any formatted context such as text, lists, or tables.

Here's how the brackets work in text:

Adjusting a Text Selection with Brackets

Four score and seven years ago.

To extend an existing selection to the right, the user makes a right bracket past the end of the selection. (A left bracket before the beginning of the selection extends to the left.)

Four score and seven years ago.

The bracket gesture acts on whole words in text, so the selection is extended to include the word under the bracket.

Four score and seven years ago.

The user makes a right bracket to shrink an existing selection from the right. (A left bracket shrinks the selection from the left.)

Four score and seven years ago.

The resulting selection includes the word under the bracket.

Four score and seven years ago.

If a right bracket is drawn over the last word in the selection, the selection will be adjusted by letter, not word.

Four score and seven years ago.

This allows the user to adjust within a word.

Here's how the brackets work in a list:

Adjusting a List Selection with Brackets

A Clockwork Orange
Planet of the Apes
China Blue
Return of the Secaucus Seven
Tremors

To extend an existing selection to include items following the selection, the user makes a right bracket.

A Clockwork Orange
Planet of the Apes
China Blue
Return of the Secaucus Seven
Tremors

The selection extends to include the item under the bracket.

A Clockwork Orange
Planet of the Apes
China Blue
Return of the Secaucus Seven
Tremors

To extend an existing selection to include items preceding the selection, the user makes a left bracket.

A Clockwork Orange
Planet of the Apes
China Blue
Return of the Secaucus Seven
Tremors

The selection extends to include the item under the bracket.

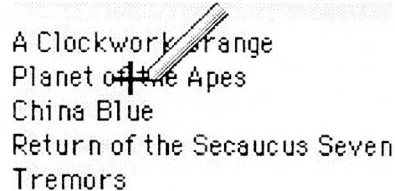
✦ Discontiguous Multiple Selections

The previous section described how to allow the user to select multiple objects that are adjacent. It is often helpful to the user to allow the selection of multiple objects that are *not* adjacent. This is referred to as **discontiguous selection**.

Discontiguous selection is commonly supported for lists, tables, and applications dealing with graphic objects.

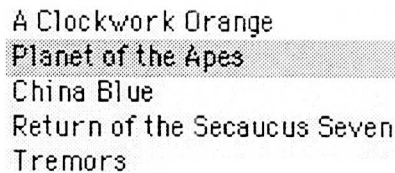
If you choose to support discontiguous selection, use the plus gesture as a toggle. A plus over an unselected object should select it, and a plus over a selected object should deselect it, as shown in the example below:

Discontiguous Selection in a List



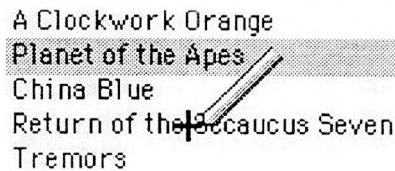
A Clockwork Orange
Planet of the Apes
China Blue
Return of the Secaucus Seven
Tremors

The user makes a plus gesture to select the desired item.



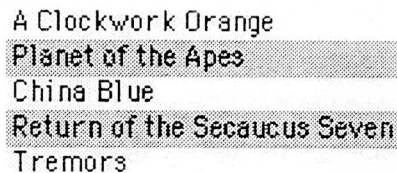
A Clockwork Orange
Planet of the Apes
China Blue
Return of the Secaucus Seven
Tremors

After the gesture is processed, the selection feedback appears.



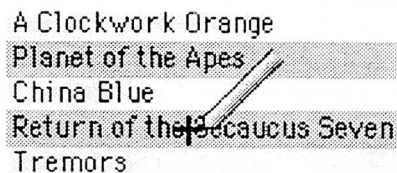
A Clockwork Orange
Planet of the Apes
China Blue
Return of the Secaucus Seven
Tremors

The user makes a plus on another item.



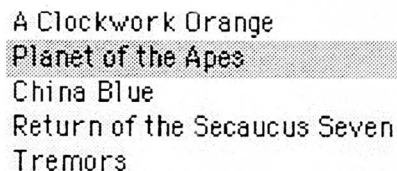
A Clockwork Orange
Planet of the Apes
China Blue
Return of the Secaucus Seven
Tremors

On pen-up, the item is selected.



A Clockwork Orange
Planet of the Apes
China Blue
Return of the Secaucus Seven
Tremors

The user makes the same gesture, the plus, to remove an item from the current selection.



A Clockwork Orange
Planet of the Apes
China Blue
Return of the Secaucus Seven
Tremors

On pen-up, the item is deselected.

Below is an example of a table with two selected columns:

1991 Sales					
	Jan.	Feb.	March	April	May
1					
2					
3					
4					
5					
6					
7					

Discontiguous Selection
in a Table

Auto-Selection of Gesture Targets

Because the pen allows the user to indicate the operand as well as the operation in one gesture, the PenPoint user need not always select the object to be operated on.

For example, the user can draw a cross-out to delete an object, a circle to edit a label, a check to bring up an object's option sheet, a circle-flick to find the next occurrence of a word, all without first selecting the object.

Often, such a gesture results in a pop-up sheet, pad, or note—an edit pad for a document in the table of contents, an option sheet for an icon, a proof menu for a word, a confirmation note for deleting a document, or the like. In all such cases you should programmatically select the object targeted by the gesture before displaying the pop-up in order to provide a visual relationship between the pop-up and its target.

Text Selection

The *de facto* standard for text handling in mouse-based user interfaces is that clicking the mouse in text sets an I-beam cursor indicating the insertion point for keyboard input, and keyboard input replaces any existing text selection.

PenPoint applications should follow these conventions in order to provide a base level of interoperability with traditional, mouse-based systems. Set an I-beam when the user taps in text, and replace any existing text selection when the user enters text using the keyboard (either the hardware keyboard or a software keyboard accessory).

Selection in Option Sheets

Because PenPoint option sheets are modeless, the user can change the selection while the option sheet is displayed. And option sheets can contain components that themselves accept the selection, such as fields or text boxes.

This presents a problem in a single-selection model: if the user makes a selection within the option sheet, the object associated with the option sheet will be deselected. When that happens, the **Apply** button on the option sheet will become inactive.

In order to solve this problem, the PenPoint operating system supports the concept of a **preserved selection**. You need to specify that the existing selection remain selected when the user makes a selection within an option sheet. That allows the option sheet's command buttons to remain active.

Here's an example that illustrates this dual selection:

Contents > Document

Title: Birds of Bali

Author: Murray Bowles

Created: 1/03/91 2:15 P.M.

Last Modified: 1/14/91 10:02 A.M.

Filed Size: 540 K

Comments:

This document was originally prepared for the mid-year conference on tropical ornithology. Later revisions were made by Rolando Aldenford,

Apply **Apply & Close**

Birds of Bali	4	
June Expense Report	5	Option Sheet with Selection
Restaurant Reviews	6	
Memo on Mice	7	

The figure shows the option sheet for a document in the table of contents. The document whose options are being shown remains selected, even though the user has selected some text in the **Comments** box.

Chapter 21 / Moving and Copying

This chapter gives guidelines for implementing the move and copy operations.

Topics covered include:

- ◆ User model.
- ◆ Detailed description of both the move and copy processes.
- ◆ Inter-application data transfer issues.
- ◆ Step-by-step examples.

▼ User Model: Drag & Drop

This section describes the PenPoint™ operating system user model for moving and copying objects, and contrasts it with the model presented by traditional, mouse-based interfaces.

▼ Move and Copy in Mouse-Based Interfaces

Mouse-based GUIs present the user with two very different models for moving and copying objects: the direct manipulation **drag & drop** model, and the clipboard-based **cut/copy/paste** model.

The drag & drop model is simple and direct: The user drags the object to the desired location with the mouse. It works well for objects such as icons on the work surface or figures in a graphics editor.

But, as traditionally used, the drag & drop method has severe limitations:

- ◆ The application designer must map it to either move or copy, since there's no way for the user to distinguish which operation is desired.
- ◆ It doesn't work for all types of objects (text, for example).
- ◆ It doesn't work when the source and destination are not in view at the same time.
- ◆ It doesn't work when the source and destination are in separate applications.
- ◆ It doesn't work for deferred or repeated operations.

Because of these limitations, traditional graphical user interfaces supplement the drag & drop method with the clipboard-based method that supports the cut, copy, and paste operations.

Move and Copy in PenPoint

The PenPoint operating system's drag & drop method addresses the limitations described in the previous section:

- ◆ It provides a way for the user to distinguish move from copy.
- ◆ It works for all types of objects: icons on the bookshelf, entries in the table of contents, text in a word processor, figures in a graphics editor, cells in a spreadsheet, and so on.
- ◆ It works when the source and destination are not in view at the same time.
- ◆ It works for transferring data across document boundaries, and between different types of documents.

To fully deliver the benefits of this unified model to the user, it is very important that all applications follow the same conventions for move and copy.

Initiating the Drag

The user initiates the drag operation either from the Edit menu or with the press (move) or tap-press (copy) gestures.

After the user initiates the operation, the system signals that the object is ready to be dragged by surrounding it with an animated marquee (single for move, double for copy). In implementation terms, what actually happens when the move or copy marquee begins is that PenPoint displays a marquee **drag icon** over the selected object.

The Drag Icon

This section describes how to display the drag icon in various situations.

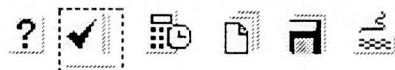
Ideally, the drag icon should contain an exact rendering of the selected object, so that it appears to the user as if the marquee simply begins around the selected object, as shown in the following examples:

Four score seven and years ago.

A Clockwork Orange

Planet of the Apes

China Blue



Drag Icons that Mimic
the Selected Object

All the application needs to do is supply the coordinates and dimensions of the region covered by the selection. The PenPoint operating system automatically copies the portion of the screen in the selected region to the icon.

⚡ Dragging Large Objects

The selected object may be too large to drag; a selected text span, for example, might well be larger than the screen itself. If the objects in your application can be so large that they are impractical to drag, you need to limit the drag icon to a manageable size. The following example shows the policy adopted by the PenPoint text component for dragging large spans of text:

He wouldn't have dared put the pills under the rug even if he thought he had time to do so before she came back -- the packages were small, but the bulges would still be all too obvious. As he heard her go into the downstairs bathroom, he took them, reached painfully around his body, and stuffed them under the mattress.

Move Marquee for Large Text Selection

In the above example, the user has selected an entire paragraph, and then chosen Move from the Edit menu. The text component has determined that the selection spans more than one line, and so has limited the move marquee to a rectangle at the beginning of the paragraph.

⚡ Dragging Offscreen Objects

The user can invoke move or copy from the menu when the selection is scrolled entirely off screen, making it impossible to copy the bits from the screen into the icon.

To support this scenario, PenPoint provides standard drag icons for both move and copy:



Standard Drag Icons for Move and Copy

When the selection is entirely off screen, the PenPoint operating system automatically displays the appropriate drag icon in the center of the screen.

▼ **Completing the Drag**

The user completes the move or copy operation by dragging the drag icon with the pen to the destination.

▼ **Drag Rectangle**

As the user drags, a dashed outline the size of the object, called the **drag rectangle**, follows the pen.

The drag rectangle, like the marquee surrounding the drag icon, is single or double to indicate move or copy.

▼ **Targeting the Destination**

When the destination allows free-form positioning of objects, as is usually the case in drawing or painting documents, the drag rectangle allows the user to position the object at the desired destination. When the user lifts the pen, the object should appear precisely where the outline was.

In such situations, the drag outline should remain in the same position relative to the pen as when the pen touched the object.

Often the destination imposes constraints on the positioning of objects—as, for example, in text, lists, and tables.

The user model for targeting the destination in such constrained contexts is to point the pen at the white space between two adjacent objects. In text, this means pointing between two words or two paragraphs. In a list, it means pointing between two items in the list. In a table, it means pointing at the border dividing two rows or columns.

A usability problem may arise here if the user is unclear whether to target by reference to the drag rectangle or the pen tip. For example, in text, should the user point the pen or the left edge of the drag rectangle at the destination?

To avoid this problem, when the user begins dragging in text, snap the drag outline so that the left edge is under the pen tip. That way, the user will succeed using either the pen tip or the drag rectangle as the reference point for targeting.

Note that because the snapping of the drag rectangle must be done when the user begins dragging away from the move or copy icon, the decision whether or not to snap the rectangle must be based on whether or not the *source* document is constrained or free-form.

► Moving and Copying Between Applications

This section mentions some implementation issues related to inter-application data transfer.

► Standard Data Types

There are four standard data types available for use by applications under PenPoint 1.0:

- ◆ **ASCII.** If your application supports the display of text in any form, you should read and write ASCII for the transfer of unformatted text.
- ◆ **RTF (Rich Text Format).** If your application supports the display of formatted text (that is, text styles such as bold, italic, underscore, and so on) then you should read and write RTF.
- ◆ **Scribble.** If your application supports the display of untranslated ink in any form you should read and write the scribble data format defined by GO.
- ◆ **PICSEG.** If your application supports the display of mixed ink, text, and graphics, it should read and write the PICSEG data format defined by GO.

► Refusing a Data Transfer

When the user drops a drag icon onto a destination that can't accept the type of data represented by the icon, the destination should refuse to accept the data.

The drag icon will jump to the location of the drop and remain floating.

The title and menu lines are examples of areas that will not accept any type of data, so users can always “park” a drag icon over them.

► Variations on Move and Copy

This section mentions a couple of variations on move and copy.

► Move as Adjust

The fact that the user can lift the pen after the press or tap-press gestures opens the door to context-specific enhancements to move and copy.

If it makes sense in your application, you can think of the move mode more broadly, as an “adjust” operation. When the user presses or tap-presses to enter move or copy mode, enhance the standard marquee to allow the user to not only drag the object as a whole but also to drag subparts to adjust a particular aspect of the object.

Mapping Move to Copy

Some objects can be copied but not moved. Often this is because the object has special semantics, as in the case of objects that serve as templates for the creation of new objects. Or it may simply be because the document containing the object is currently write-protected, and therefore cannot be altered in any way.

In such cases (unless there is a particular reason why doing so would cause user confusion), you should map the press gesture, which ordinarily initiates a move, to copy. The mapping will be apparent to the user because the marquee and drag box will be double rather than single.

This allows the uninitiated user who tries to move the object by doing a press-drag to at least accomplish half the intended operation: The object appears at the destination even if it is not deleted from the source. It also benefits the user who knows that the object can't be moved, by making the copy a bit easier (press instead of tap-press).

Examples

The next several pages give step-by-step examples illustrating moving and copying in several common scenarios. The first example shows a word being moved to a different position in a sentence. Note that the left edge of the drag rectangle jumps to the pen tip to aid in targeting.

EXAMPLE 1 Moving a Word in Text

Four score **seven** and years ago.

The user first selects the word to be moved.

Four score **seven** and years ago.

Then the user initiates the move with the press-hold gesture.

Four score **seven** and years ago.

When the user drags the pen, the marquee outline jumps so that the pen tip is centered on the left edge of the outline.

Four score **seven** and **years** ago.

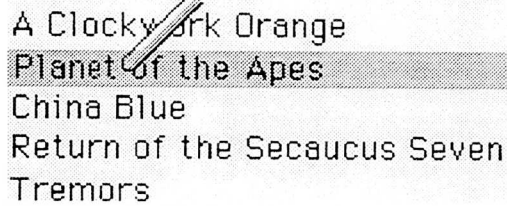
The user points the pen tip directly at the desired location—in the gap between two words.

Four score and **seven** years ago.

When the user lifts the pen, the move is completed and the text reformatted. The newly moved word remains selected.

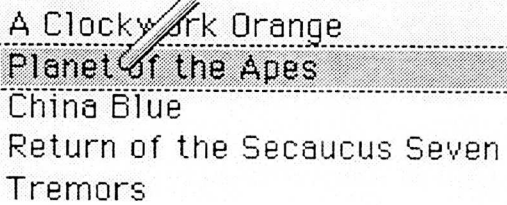
The next example shows an item being moved to a new location in a list.

EXAMPLE 2
Moving an Item in a List



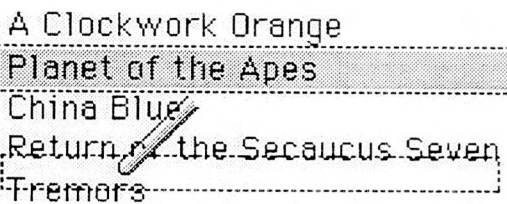
A Clockwork Orange
Planet of the Apes
China Blue
Return of the Secaucus Seven
Tremors

The user first selects the item to be moved.



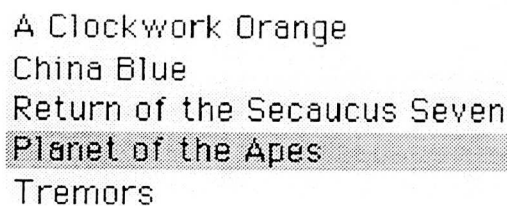
A Clockwork Orange
Planet of the Apes
China Blue
Return of the Secaucus Seven
Tremors

Then the user initiates the move with the press gesture.



A Clockwork Orange
Planet of the Apes
China Blue
Return of the Secaucus Seven
Tremors

The user drags out the dashed outline until the pen tip points between the rows where the item is to be moved.



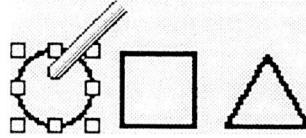
A Clockwork Orange
China Blue
Return of the Secaucus Seven
Planet of the Apes
Tremors

When the user lifts the pen, the move is completed and the list redisplayed.

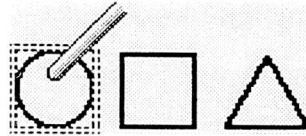
The newly moved item remains selected.

Graphics applications can drag either the standard drag rectangle or a rendition of the object itself.

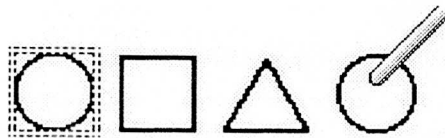
EXAMPLE 3
Copying a Figure



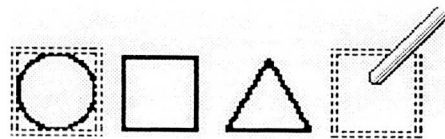
The user first selects the figure to be moved.



The user initiates the copy with the tap-press gesture.

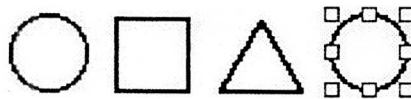


When the user drags the pen, the application can drag a rendition of the object itself.



Alternatively, the application can drag the standard drag box.

If the drag box is used, it should *not* jump relative to the pen. It should also be the exact size of the selection. This allows the user to use the outline as a guide for positioning the object precisely.

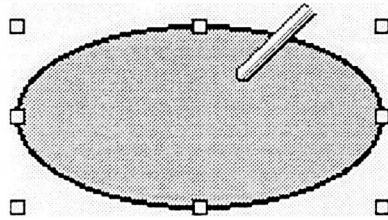


When the user lifts the pen, the copy is completed.

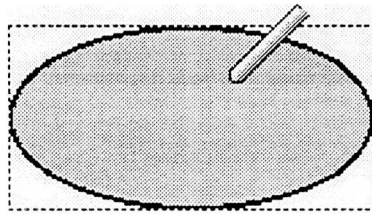
The newly copied object remains selected.

The final example shows the use of the standard move icon for moving a figure to an off-screen destination.

EXAMPLE 4 Moving a Figure to a Far Destination

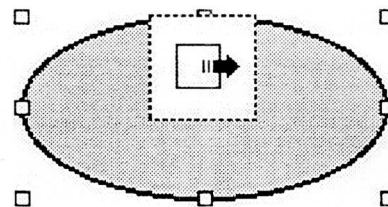


The user first selects the figure (or figures) to be moved.



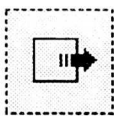
Then the user touches the selected figure with the pen and pauses until the move marquee appears.

(At this point, the user can drag to a local destination without lifting the pen, as described in the previous figure.)

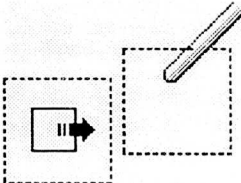
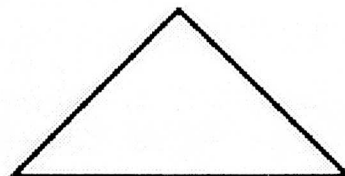


The user lifts the pen at this point, causing the move icon to appear.

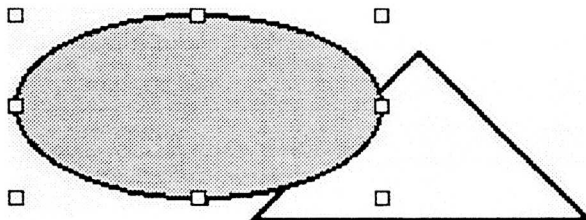
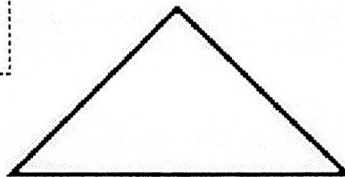
The move icon has a picture symbolizing the move operation, and is surrounded by the move marquee. It is always the same size, regardless of the size of the selected object.



While the move icon is displayed, the user brings the destination into view by turning to another document (or scrolling within the same document).



When the destination is in view, the user continues the drag by touching the move icon and dragging out a dashed outline.



When the user lifts the pen, the data transfer takes place, and the object appears at the destination.

The object is placed in the same location relative to the pen as when the move was initiated.

Appendix A / PenPoint Design Checklist

Here is a checklist of important things your application should do or include.

- ☐ **Basic application architecture.** Is your application a document, accessory, or background service?
- ☐ **Dual command path.** As far as possible, all commands should be available from both visible controls (menus, palette lines, option sheets, dialog sheets) and gestures.
- ☐ **Core gesture mode.** Your application should have a mode in which all the core gestures work as expected. It is alright to have other modes.
- ☐ **Modes.** If your application has input modes, you should have a visible mode control and observe the toggle mode gesture (left-right).
- ☐ **Option Sheets.** Use the option sheet facility for as much of the functionality of your application as appropriate.
- ☐ **Scalability.** Your application should respond gracefully to small screens (320 x 200 pixels) or to the user's changing the system font. It should always use layout units for laying out visual elements.
- ☐ **Quick Help.** Every control and every region of your application's client area should have a Quick Help message.
- ☐ **Help Notebook.** During installation, your application should add a section to the Help notebook with task-oriented help for your application.
- ☐ **Stationery.** During installation, your application should add a section to the Stationery notebook with stationery as appropriate.
- ☐ **Icons.** You should have two icons (normal and large) defined in your application's resource file.
- ☐ **Paper-like visuals.** If your application is a document or notebook, all the visuals should be paper-like.
- ☐ **Support for embedding.** Your application should allow documents to be embedded within it.

-  **Marks for Reference Buttons.** Your application should allow the user to select within the document and make a reference button linked to that selection that scrolls into view when tapped.
-  **Move and Copy.** Your application should support transfer of the standard data types—ASCII, RTF, Scribble, and PICSEG—as appropriate.

PENPOINT USER INTERFACE DESIGN REFERENCE INDEX

- 3-D icons, 217
- About Application sheet, 61
 - customizing, 194
- About command (Document menu), 57, 61
- About Contents command (table of contents Document menu), 85
- About Document sheet, 61
- Accessories, 163
 - icon, 13, 74
 - application, 217
 - palette, icons in, 73, 163, 217
- Accessory
 - icons, 77
 - instances, 163
 - keyboard, 202, 273
- Access sheet, 67, 201
 - Access Speed control, 67
 - active documents and, 102
 - customizing, 201
- Address Book icon, 77
- Align commands (MiniNote Arrange menu), 137
- Ancestors, gestures and, 233
- Application
 - architecture, 295
 - functionality, 223
 - basic guidelines, 224
 - dividing, 225–236
 - state, 222
 - stationary, 120
 - see also* Applications
- Application Framework
 - document options, 44
 - embedding and, 152
 - standard menus, 192
- Application icons, 73, 78, 216–222
 - for accessories, 217
 - bitmaps and, 217–218
 - design guidelines for, 219–222
 - bitmap relationships, 220
 - icon mask design, 221–221
 - no “3-D” style, 221
 - simplicity, 219
 - size, 219
 - dimensions and hot point for, 217–219
 - for documents, 216–217
 - GO’s conventions for, 216–217
 - using, to show state, 222
- Applications
 - background, 159
 - bundled, 127–142
 - busy clock display and, 81
 - configurability of, 224
 - data types for, 289
 - designing, for pen, 152
 - document, 157–158
 - dynamic behavior of option sheets and, 203
 - embedded objects, 166
 - embedding support and, 166
 - installing, 162–165
 - modal, 253
 - moving and copying between, 289
 - non-document, 159
 - Quick Help for, 215
 - types of, 159
- Applications page (Installed Software section), 98
 - views, 109
 - option menu, 111
- Apply button
 - clean and dirty controls and, 46–47
 - option sheets and, 44
- Arrange menu (MiniNote), 137
- Arrows & Drag Box scroll margins, 263–264
- Arrows, for scroll margin, 36
- Arrows gesture family, 24
- Arrow (Up and Down) gesture, 25
 - Double, 266
 - guidelines for, 240
 - MiniText, 133
 - for zooming, 266
- ASCII format, 289
- Audible feedback, 217–218
 - default, 213
 - for minor errors, 213–214
- Auto Power-Off Devices (Power preferences), 96
- Auto-selection, 283
- Auto Shutdown (Power preferences), 96, 97
- Auto Suspend (Power preferences), 96, 97
- Background applications, 159
- Baselines, for icon bitmaps, 218
- B gesture, 27
 - in gesture mode, 259
 - in MiniNote, 142
 - in MiniText, 134
- Bitmap editor icon, 77, 216
- Bitmaps
 - for icons, 217–218
 - small and large, 220
- Boarders
 - button, 177
 - for embedded documents, 19–20
- Bold buttons, 28, 177
- Bold style, 16
- Bookshelf, 13–14
 - disk-based, 13–14
 - icon option sheet and, 75
 - icons, 13, 74
 - closed and open states of, 221–222
 - list of, 76
 - illustrated, 12
 - main, 13
 - reference buttons on, 20, 171
 - targeting policy, 233
 - view, 109
 - Layout sheet, 112
- Bounding box, for drag feedback, 280
- Boxed lists, 32
 - checklist vs., 182–183
 - illustrated, 32
 - multiple, 184
 - choice, 32
 - modes, 248
 - palette line with, 39
- Boxed pads, 50
 - buttons, 50
 - writing, 48
 - embedded, 49
 - pop-up, 48
 - using, 51
- Brackets gesture, 25
 - adjusting list selection with, 281
 - adjusting text selection with, 281
 - extending selection with, 280–281
 - family, 24
 - in gesture mode, 259
 - guidelines for, 238
 - hot point for, 231
 - MiniNote, 141
 - see also* Gestures
- Bring To command (table of contents View menu), 86
- Bundled applications, 127–142
- Busy clock display, 81
- Buttons, 28–29
 - command, 178
 - defined, 179
 - non-standard, 179
 - standard modal, 178–179
 - standard modeless, 178
 - customizing, 177–178
 - labels for, 175
 - deactivating, 175
 - wording, 210–211
 - non-command, 179

- on boxed pads, 50
 - styles for, 175–178
 - bold, 177
 - half-outlined, 177
 - outlined, 176
 - raised, 176
 - square, 176
 - types of, 28–29
 - in user interface, 151
 - see also* Reference buttons
-
- Calculators, 13
 - Cancel button
 - command, 178–179
 - for progress notes, 211
 - Capitalization convention
 - in button labels, 175
 - for messages, 210
 - Capital letter gestures, 27
 - collisions and, 233–234
 - hot point for, 231
 - see also* Gesture accelerators
 - Caret gesture, 16, 24
 - for Create menu, 87
 - family, 23–24
 - summary, 235
 - in gesture mode, 258
 - guidelines for, 237
 - MiniNote, 141
 - tab stops and, 132
 - see also* Gestures
 - Caret-Tap gesture, 25
 - guidelines for, 239
 - MiniNote, 140
 - MiniText, 133
 - see also* Gestures
 - Cells, reference button labels for, 172
 - Center command (MiniNote Arrange menu), 137
 - C gesture, 27, 56
 - Character lists template, 242
 - Character option sheet (MiniText), 131
 - Character page, 46
 - Character recognition engine, 234
 - Checkboxes, 32
 - multiple, 184–185
 - for scrolling multiple checklists, 187
 - Check gesture, 16, 24
 - Connected Disks page, 115
 - in gesture mode, 258
 - guidelines for, 237
 - hot point for, 231
 - MiniNote, 135, 141
 - on document title, 89
 - option sheets response to, 203
 - over icons, 75
 - response to, 46
 - see also* Gestures
 - Checklists, 29
 - alternative
 - boxed lists, 32
 - toggle switch, 30
 - boxed lists vs., 182–183
 - design, 295–296
 - with fields, 186
 - menus with, 41–42
 - multiple, 31, 184
 - comparison with single, 31
 - scrolling, 187
 - pop-up, 30
 - scrolling, 196
 - scrolling, 196
 - Checkmark, 182
 - gesture mode picture, 248
 - in zero or one style, 185
 - Checkpoint command (Document menu), 57, 193
 - Checks gesture family, 24
 - summary, 235
 - see also* Check gesture; Check-Tap gesture
 - Check-Tap gesture, 25
 - guidelines for, 239
 - Circle-Flick-Down gesture, 25
 - Circle-Flick gesture (Up and Down), 133
 - Circle-Flick-Up gesture, 25
 - Circle gesture, 16, 24
 - editing pads and, 189
 - family, 24, 235
 - gesture mode and, 257, 258, 259
 - guidelines for, 237
 - hot point for, 231–232
 - MiniNote gesture mode, 135, 141–142
 - MiniNote ink mode, 135
 - MiniText, 133
 - see also* Gestures
 - Circle-Line gesture, 25
 - gesture mode and, 257, 259
 - guidelines for, 239
 - Circle-Tap gesture, 25
 - gesture mode and, 257, 259
 - guidelines for, 239
 - Clear command (MiniNote Edit menu), 136
 - Clock, 13
 - busy, 81
 - icon, 77
 - Closed icon, 74, 221–222
 - Collapse command (table of contents View menu), 86
 - Collisions, gesture, 233–234
 - Command buttons, 178–180
 - defined, 179
 - modeless edit pads and, 190–191
 - non-standard, 179
 - standard modal, 178–179
 - standard modeless, 178
 - see also* Command line
 - Command line, 178, 179
 - buttons for, 179
 - proper use of, 179
 - see also* Command buttons
 - Commands
 - bold buttons and, 177
 - button labels and, 175
 - Document menu, 57
 - default, 193–194
 - Edit menu, 62–65
 - customized, 195
 - default, 194–195
 - fundamental, 226
 - layering, 224
 - menus with, 41
 - names of, 226
 - table of contents
 - Create menu, 86–87
 - Document menu, 85
 - Edit menu, 86
 - View menu, 86
 - two-state switches and, 198
 - Comments sheet, 67, 202
 - customizing, 202
 - Completion note, 211
 - timing-triggered, 212–213
 - Compound documents, 18
 - illustrated, 18
 - Configuration
 - modification, 164
 - options, 165
 - user, of application, 224
 - Confirmation notes, 212
 - Connected Disks page (Connections notebook), 106–113
 - clash notes and, 71
 - Create Directory command, 195
 - disk representation in, 73
 - edit pads and, 47
 - icons and, 106
 - menus, 107
 - option sheets, 111–112
 - icon, 75
 - options menu, 111
 - Quick Installer, 113
 - controls, 112
 - views, 108–110, 113
 - Applications view, 110
 - Bookshelf view, 109
 - Directory view, 108
 - Fonts view, 110
 - installable software view, 109–110

- Connected Printers page (Connections notebook), 116–118
 - illustrated, 116
 - menus, 116–117
 - options sheets, 117–118
- Connections icon, 13, 76
- Connections notebook, 105–119
 - Attached Printers page, 163
 - Connected Disks page, 106–113
 - clash notes and, 71
 - Create Directory command, 195
 - disk representation in, 73
 - edit pads and, 47
 - icon option sheet and, 75
 - views, 108–110
 - Connected Printers page, 57, 116–118
 - Network View page (Disks section), 114–115
 - Network View page (Printers section), 118–119
 - overview, 105
 - table of contents, 105
 - see also* Help notebook; Notebook; Settings notebook; Stationary notebook
- Consistency, 154
 - departure from standards and, 227
 - toggle gesture, 247
 - in UI design, 149
- Constraining translation, 241–243
 - ambiguity and, 241–242
 - with dictionary and templates, 242–243
 - to letters, numbers, punctuation, symbols, 242
- Constraints, 241
 - context-specific, 242
 - guidelines for using, 243
- Containers, 40
- Control buttons, 28, 175–180
- Control margins, 184
- Controls, 28–39, 175–191
 - clean and dirty, 46–47
 - customized, 28
 - deactivating vs. hiding, 209
 - dialog sheet, 205
 - edit pads, 189–191
 - for embedded documents, 20
 - gauges, 38
 - grouping, 208
 - layout, 196
 - option and dialog sheets, 206–209
 - lists, 29–32, 181–188
 - in menus, 41–42
 - message line, 214
 - mode switch and, 251
 - option sheet, 56
 - palette lines, 39
 - scroll margins, 35–37
 - separate groups of, 197
 - for showing and hiding, 224
 - subpage, 38
 - text fields, 32–34, 188
 - toggle switches, 180–181
 - for zooming, 266–267
- Controls sheet, 60, 66, 201
 - Boarders control, 60
 - customizing, 201
 - gesture margin and, 257
 - message line and, 214
 - mode switch option, 251
 - Show switch, 66
- Copy command (Edit menu), 62
- Copy function, 16
- Copying
 - between applications, 289
 - document with same section, 71
 - drag & drop gesture for, 68
 - figure, 293
 - importing/exporting and, 70
 - in mouse-based interfaces, 285
 - in PenPoint, 286
 - variations, 289–290
 - see also* Moving
- Copy marquee, 68
- Core gestures, 16–17, 24–25
 - application functionality and, 224
 - collisions and, 234
 - consistency and, 154
 - defined, 236
 - for gesture mode, 257, 258
 - guidelines for, 236–237
 - MiniText, 133
 - mode, 247
 - design checklist and, 295
 - variations, 236
 - guidelines for, 239
 - see also* Gestures
- Cork margin, 20, 56
 - of Help notebook index, 125
 - hiding, 56
 - icons, 74
 - option sheet and, 75
 - illustrated, 55
 - reference buttons and, 172
- Corners gesture family, 24
 - hot point for, 231
 - summary, 236
 - see also* Gestures
- Create command (Edit menu), 195
- Create menu, 20
 - caret gesture family and, 233
 - Connected Printers page, 117
 - system-wide, 120
 - table of contents, 86–87, 195
- Creating, embedded documents, 20
- Cross Out gesture, 16, 24
 - default targets, 232
 - in gesture mode, 258
 - guidelines for, 237
 - hot point for, 231
 - MiniNote gesture mode, 135, 141
 - MiniNote ink mode, 135
 - tab stops and, 132
 - see also* Gestures
- Cursor, I-beam, 283
- Customize button (Installed Handwriting page), 100
- Customizing
 - buttons, 177–178
 - edit pads, 189
 - option sheets, 201–202

- Data structures, designing, 166
- Data transfer
 - design checklist and, 296
 - refusing, 289
- Date preferences (Settings notebook), 94
- Deactivating, hiding controls vs., 209
- Default document
 - icon, 77
 - option sheets, 44
- Default menus, 56
 - Document menu, 57–61
 - Edit menu, 62–65
 - Options menu, 44, 66–67
- Default objects, 46
 - determining, 203
- Default PenPoint application, 17
- Delete command (Edit menu), 62
- Delete command (MiniNote Edit menu), 136
- Delete function, 16
- Deselecting, 279
 - an existing selection, 280
- Design
 - checklist, 295–296
 - graphic, 149
 - modular, 152–153
 - user-centered, 149
 - wording in, 150
- Designing
 - application icons, 216–222
 - guidelines for, 219–221
 - masks for, 221–222
 - consistency and, 154
 - data structures, 166
 - editing pads, 189
 - for Embedded Document Architecture, 152–153
 - handwriting translators, 241–243
 - menu line, 226
 - for notebook metaphor, 151
 - for the pen, 152
 - for pen-based mobile computing, 151–154

- for scalability, 153–154
- user models and, 245–246
- Development
 - iterative, 150
 - milestones, 5
- Devices, comparison, 10
- Dialog sheets, 40, 162, 205–206
 - checklists in, 184
 - command buttons, 178
 - for installable accessory instances, 163
 - for installable stationary, 162
 - layout guidelines, 206–209
 - deactivating vs. hiding controls, 209
 - non-standard layout, 206
 - pop-up vs. in-line, 207–208
 - standard layout, 206
 - modal, 165, 205–206
 - modeless, 205
 - advantages of, 205
 - multi-page, 43
 - multiple, 205
 - pop-up lists, 183
 - Quick Help and, 215
 - single, 205
 - titles, 205
 - uses, 205
 - vertical checklists, 182
- Dictionary
 - icon, 78
 - installable software views and, 109
 - installed, 105
 - template, 242
- Direct manipulation, 150
- Directories
 - merge, 72
 - name clash and, 72
- Directory icon, 108
- Directory view (Connections Disk page), 108
 - Layout sheet, 111
 - Options menu, 111
- Discontiguous selection, 282–283
- Disk
 - Bookshelf, 109
 - browsing, 105
 - Connected, 106–113
 - icons for, 106
 - connection, 115
 - directory icons, 73
 - icons, 73, 77
 - contents, 78
 - management, 105
 - menu line for, 107
 - network, 114–115
 - option sheet, 112
 - quick install, 113
- Disk-based bookshelf, 13–14
 - exposing, 14
- Disk-based operating system, 10–11
- Display
 - embedded objects and, 166–167
 - parameters, 274
 - size, 153
- Display option sheet (MiniText), 132
- Document menu, 57–61
 - About command, 61
 - default, 193
 - default commands, 56–57
 - About, 194
 - Checkpoint, 193
 - Print Preview, 193–194
 - Print Setup, 193–194
 - Revert, 193
 - Embedded Printing sheet, 60–61
 - guidelines for using, 192
 - Headers & Footers sheet, 59
 - Print Layout sheet, 58
 - Print sheet, 57
 - table of contents, 85
 - uses, 192
- Document outline, 218–219
- Documents, 14
 - applications as, 157–158
 - basic layout for, 55
 - components for, 56
 - illustrated, 55
 - compound, 18
 - cork margin, 20
 - creating new, 86–87
 - default menus for, 56
 - disk Bookshelf, 109
 - email, 121
 - embedded, 17
 - Apply To control, 60
 - boarders and controls for, 19–20
 - creating, 20
 - icons and, 73
 - in-line, 169–170
 - pop-up, 169–171
 - printing, 60–61
 - special handling of, 168–169
 - up-down gesture and, 66
 - fax, 121, 165
 - help, 124
 - icons for, 73
 - application, 216–217
 - list of, 77
 - In box, 13
 - options for, 164
 - incoming, 121
 - marking locations within, 171–172
 - Menu box (Stationary notebook), 120
 - multi-page, 157–158
 - multiple, table of contents and, 86
 - name clash notes and, 71
 - notebook page and, 157
 - option sheets for, 89
 - organizing, 14
 - Out box, 13
 - outgoing, 122
 - page-oriented, 157–158
 - printing
 - layout for, 58
 - settings for, 57
 - receiving, 159
 - reference buttons and, 20, 171
 - resize handles for, 272
 - sending, 159
 - special handling of, 168–169
 - stationary, 120
- Document stationary, 162
- Document title line, 47
- Document-wide options, 44
 - adding default, 200–202
 - adding new, 202
- Double-Arrow (Up and Down), 266
- Double-Caret gesture, 23, 238
- Double-Circle gesture, 25
 - in gesture mode, 259
 - guidelines for, 238
- Double-Down Arrow gesture, 25
- Double-Flick, 25
 - guidelines for, 239
 - ink editing gesture, 259
 - MiniNote, 141
 - MiniText, 133
- Double-Left Arrow gesture, 25
- Double-line convention, 31
 - checklists and, 184
 - for scrolling multiple lists, 187
- Double-Right Arrow gesture, 25
- Double-Tap, 25
 - in gesture mode, 259
 - guidelines for using, 239
 - MiniNote, 140
 - MiniText, 133
 - zooming and, 266
- Double-Up Arrow gesture, 25
- Down-Left-Flick gesture, 26, 240
 - MiniText, 133
- Down-Left gesture, 26, 240
 - MiniNote, 141
 - MiniText, 133
- Down-Right-Flick gesture, 26, 240
 - MiniText, 133
- Down-Right gesture, 17, 24
 - in gesture mode, 258
 - guidelines, 237
 - MiniNote, 141
- Drag & drop interface, 68–72
 - consistency and, 154
 - example, 69
 - import/export and, 69–70
 - limitations, 285
 - mouse-based, 285
 - name clash notes and, 71–72

- operation, 68–69
 - cancelling, 69
 - PenPoint, 286
 - user model, 285–286
 - uses, 68
 - Drag
 - box, 263–264
 - completing, 288
 - feedback, 280
 - handle, 36
 - icon, 286–287
 - refusing, 289
 - standard, 287
 - initiating, 286–287
 - outline, 288
 - rectangle, 288
 - snapping, 288
 - Dragging
 - large objects, 287
 - offscreen objects, 287
 - Dual command path, 15–16
 - for application functionality, 224
 - boarders and, 20
 - design checklist and, 295
 - edit pad design and, 189
 - for switching modes, 246
 - toggle gesture and, 247
 - for zooming, 265
-
- Edit function, 16
 - Editing, 16
 - fill-in fields, 34
 - Help notebook index entries, 125
 - ink, 255
 - guidelines for, 255–259
 - overwrite fields and, 33, 34
 - scrolling lists, 187
 - see also* Edit pads
 - Edit menu, 62–65, 194–195
 - application-specific, 195
 - creating object guidelines, 195
 - customized, 195
 - default, 194
 - default commands, 56, 62–65, 194–194
 - Undo, 194
 - drag & drop operation and, 68
 - editing operations and, 16
 - Find command, 62, 63
 - Find sheet, 62
 - guidelines for using, 192
 - for initiating drag, 286
 - MiniNote, 136–137
 - Proof sheet, 65
 - Spell sheet, 64
 - table of contents, 86
 - uses, 192
 - Edit pads, 189–191
 - customizing, 189
 - designing, 189
 - input behavior, 190–191
 - invoking, 189
 - keyboard focus of, 189
 - modeless, 190–191
 - pop-up, 47, 189–190
 - see also* Editing
 - Ellipse convention, button labels, 175
 - Email
 - documents, 121
 - service, 164
 - Embedded chart, 167
 - Embedded Document Architecture (EDA), 17–20
 - designing for, 152–153
 - marking document locations and, 171–172
 - Embedded documents, 17
 - Apply To control (Embedded Printing sheet), 60
 - boarders and controls for, 19–20
 - compound documents and, 18
 - creating, 20
 - design checklist and, 295
 - icons and, 73
 - illustrated, 18
 - in-line, 169–170
 - pop-up, 169–171
 - printing, 60–61
 - Location control, 60
 - special handling of, 168–169
 - up-down gesture and, 66
 - see also* Documents
 - Embedded icons, 168
 - in-line, 168
 - pop-up, 170–171
 - see also* Icons
 - Embedded objects, 152
 - displaying, 166–167
 - layout of, 168–171
 - in-line opening, 169–171
 - open document handling, 168–169
 - managing, 166–172
 - basic support for, 166–167
 - intelligent layout, 168–171
 - marking document locations and, 171–172
 - storing, 166
 - tracking, 166
 - see also* Objects
 - Embedded Printing sheet, 60–61
 - Apply To control, 60
 - Location control, 60
 - Print control, 60
 - Embedded writing pads, 49
 - pop-up pad comparison with, 49
 - resize handles for, 272
 - Embedding protocol, 166
 - spreadsheet example, 166–167
 - Error notes, 216
 - Quick Help and, 215
 - timing-triggered, 212–213
 - Expand command (table of contents View menu), 86
 - Explicit modes, 244–245
 - Exporting, 69
 - Connections notebook and, 105
 - documents, 108
 - in PenPoint, 69
 - Export note, 70
-
- Fax
 - documents, 121, 165
 - service, 164–165
 - Feedback
 - audible, 213–214
 - message line and, 214
 - selection, 187, 278–279
 - visual, 247
 - F gesture, 27, 63
 - MiniNote, 142
 - MiniText, 134
 - Fields, menu, 199
 - see also* Fill-in fields; Overwrite fields
 - File folder, directory icon, 108
 - File icon, 78, 108
 - Fill-in fields, 33
 - checklists with, 186
 - editing gestures for, 34
 - fonts, 188
 - guidelines for using, 188
 - menus with, 41
 - edit pad, 199
 - scaling, 268
 - size of, 188
 - toggle switches and, 180–181
 - visual segmentation cues and, 241
 - see also* Overwrite fields
 - Find & Replace sheet, 63
 - Find button, 63
 - Replace button, 64
 - Search button, 63
 - Find command (Edit menu), 62
 - mark protocol and, 172
 - Find sheet, 62
 - modifying and replacing, 194
 - Flick gesture, 16, 24
 - Double, Triple, Quadruple, 25
 - guidelines for, 239
 - family, 24
 - summary, 235
 - in gesture mode, 258
 - guidelines for, 237
 - hot point for, 231–232
 - MiniNote gesture mode, 135, 141
 - MiniNote ink mode, 135

- MiniText, 133–134
 - pop-up checklists and, 30
 - scrolling with, 264–265
 - to switch modes, 249
 - Tap & Flick and, 37
 - see also* Gestures
 - Flick Left-Right gesture, 26
 - guidelines for, 238
 - ink editing, 258
 - MiniNote, 140
 - to toggle modes, 256
 - Flick Up-Down gesture, 26, 66
 - in gesture mode, 259
 - guidelines for, 238
 - Float & Zoom preferences (Settings notebook), 94
 - Floating
 - navigation control area, 267
 - page, 15
 - Floppy disk icon, 77
 - Folder icon (open/closed), 78
 - Font Field preference, 188
 - Font icon, 78
 - Fonts & Layout preferences (Settings notebook), 93
 - Fonts, 101
 - control of (MiniNote), 138
 - installable software views and, 109
 - installed, 101
 - system font
 - scaling with, 269–270
 - size of, 274
 - Font size
 - system, 269–270
 - user defined, 271
 - Font size preference, 188
-
- Gauges, 38
 - Gesture accelerators, 27
 - application functionality and, 224
 - for scrolling lists, 188
 - Gesture collisions, 233–234
 - Gesture icon, 78
 - Gesture margin, 135–136, 257
 - Gesture mode, 245–246
 - core, 247
 - guidelines for editing in, 257
 - MiniNote, 135
 - toggle switch, 30
 - default, 248
 - see also* Gestures
 - Gesture recognition engine, 233
 - Gestures, 15–17, 23–27
 - bounding box, 231
 - categories of, 23
 - capital letter accelerators, 27
 - core, 16–17, 24–25
 - non-core, 23, 25–27
 - collisions of, 233–234
 - for drag & drop operation, 68
 - dual command path and, 15–16
 - families of, 23–24, 235–236
 - defined, 235
 - in fields, 34
 - functions of, 16–17
 - hot points of, 231–232
 - icons response to, 74
 - ink editing, 258–259
 - installable software views and, 109
 - MiniNote, 140–142
 - gesture margins and, 135–136
 - MiniText, 133–134
 - passing on unused, 233
 - processing, 231–240
 - for reference buttons, 79
 - selection, 187, 279
 - targeting, 231–233
 - auto-selection and, 283
 - guidelines, 232–233
 - hot points, 231–232
 - usage guidelines for, 236–240
 - use of, 224
 - for zooming, 266
 - see also specific gestures and types of gestures*
 - GO Fax icons, 77, 216
 - Graphic design
 - fundamentals, 149
 - of icons, 219–222
 - Grid
 - for aligning controls, 206
 - building translators with, 241
 - use in graphic design, 149
 - Group command (MiniNote Arrange menu), 137
 - GUI (Graphical User Interface), 149
 - button label conventions, 175
 - drag & drop model, 285
 - explicit modality and, 244–245
 - spacial modality and, 244
-
- Half-outlined buttons, 177
 - Handles, resize, 272
 - Hand preference, 35
 - Handwriting
 - engine, 100
 - icon, 78
 - installable software views and, 109
 - installed, 99–100
 - processing, 241–243
 - constraining translation and, 241–243
 - using translators and, 241
 - profile, 100
 - translators, 241
 - building, 242
 - letters, numbers, punctuation, symbols, 242
 - Handwriting Preference sheet, 188
 - Hard disk icon, 77
 - Hardware requirements, 5
 - Headers and Footers sheet, 59
 - Help, 215
 - icon, 13, 76, 123, 124
 - message, 123
 - notebook, 13, 215
 - Quick Help, 215
 - Help facilities, 123–125
 - Help notebook, 124–125
 - Quick Help, 123
 - Help notebook, 124–125, 215
 - design checklist and, 295
 - index, 125
 - table of contents, 124
 - see also* Connections notebook; Notebook; Settings notebook; Stationary notebook
 - H gesture, 27
 - Hierarchical menus, 199
 - Horizontal scrolling, 274–275
 - Hot points
 - gesture, 231–232
 - icon, 74, 217–219
 - correct placement of, 217–219
-
- I-beam cursor, 283
 - Icon bitmap, 217–218
 - specifications, 221
 - Icon mask, 221–222
 - Icons, 73–78
 - 3-D, 221
 - accessory, 77
 - application, 216–222
 - for accessories, 217
 - for documents, 216–217
 - Bookshelf, 76
 - check gesture over, 75
 - closed, 74, 221–222
 - design checklist and, 295
 - design guidelines for, 219–222
 - icon mask design, 221–222
 - no “3-D” style, 221
 - relationship of bitmaps, 220
 - simplicity, 219
 - size, 219
 - dimensions for, 217–219
 - directory, 108
 - disk, 77
 - contents, 78
 - document, 77
 - drag, 286–287

- for move and copy, 287
 - refusing, 289
 - embedded, 168
 - in-line, 168
 - pop-up, 170–171
 - embedded documents and, 73
 - embedding and, 167
 - file, 78, 108
 - gesture response of, 74
 - GO's conventions for, 216–217
 - hot points for, 74, 217–219
 - installable software, 78
 - large, 220
 - Notebook, 76
 - open, 74, 221–222
 - option sheets for, 75
 - in-line option, 171
 - size specification, 217
 - PenPoint standard list of, 76–78
 - selection feedback for, 279
 - showing application state with, 222
 - sizes of, 74, 217
 - small, 220
 - usage, 73
 - see also* Pictures
- I gesture, 27
- MiniText, 134
- Importing, 69
- Connections notebook and, 105
 - files, 13
 - Directory view for, 108
- Import note, 70
- In box, 121
- icon, 13, 76
 - state of application and, 222
 - services, 164
 - table of contents, 121
 - transfer service and, 159–161
- Index
- Help notebook, 125
 - reference buttons for, 171
- Initial View control (Disk option sheet), 112, 113
- Ink
- data object, 254
 - editing, 255
 - guidelines for, 255–259
 - functionality, 254
 - gestures, 258–259
 - using, 254–259
 - ways of, 254–255
- Ink component, 135
- gestures accepted by, 140–142
 - Notepaper, 255
 - option sheets, 138–139
 - scaling, 269–270
- Ink mode, 245–246, 255–256
- Flick Left-Right gesture, 256
- gestures, 256, 258
- over the selection, 256
- MiniNote, 135
- pen styles, 256
- selecting in, 255–256
- toggle switch, 248
- In-line style (embedded document), 168–170
- dialog/option sheets and, 207–208
 - mixing with pop-up style, 207–208
- Input modes, 244–253
- guidelines for using, 246–248
 - core gesture mode, 247
 - mode switch, 247
 - mode toggle switch, 247–248
 - modal applications, 253
 - role of, 244–246
 - switch location, 251–252
 - switch presentation, 248–250
 - user models for, 245–246
 - see also* Modes
- Insert function, 16
- Insert Line command (MiniNote Edit menu), 136
- Insert space function, 17
- Installable software icons, 78
- Installation
- dialog, 113
 - configuration, 164
 - options, 165
 - document options and, 164
 - flexibility vs. simplicity, 165
- Install button (Settings notebook), 99
- Installed Software section (Settings notebook), 13, 90, 97–101
- Applications page, 98
 - Installed Dictionaries page, 100
 - Installed Fonts page, 101
 - Installed Handwriting page, 99–100
 - Installed User Profiles page, 101
 - install sheet, 99
 - menus, 98
 - options, 98
 - overview, 97–98
 - uses, 97–98
- Installing
- applications, 13, 162–165
 - accessories, 163
 - document stationary, 162
 - flexibility vs. simplicity, 165
 - services, 163–165
 - from disk, 105
- Install sheet, 99
- Instances, accessory, 163
- Internal disk size, 103
- Italic style, 16
- Iterative development, 151
-
- Keyboard
- accessory
 - option sheet, 202
 - scales on resize, 273
 - focus of edit pad, 189
 - icon, 13, 76
 - text selection and, 283
 - typing simulation, 13
-
- Labels
- bold, 177
 - button, 175
 - wording for, 210–211
 - for dialog and option sheets, 206
 - edit pads and, 47
 - for gauges, 38
 - icon, 74
 - menu, 42, 177
 - for current mode, 249
 - pop-up lists, 183
 - reference button, 172
 - suppressing, 183
 - for Undo Edit command, 194
- Landscape orientation, 276–277
- Large icons, 74
- Layout
- automatic, 153–154
 - controls, 196
 - option and dialog sheets, 206–209
 - embedded object, 168–171
 - open document forms and, 169–171
 - open document handling and, 168–169
 - landscape orientation, 276–277
 - menu, 196–197
 - multi-column, 196
 - multiple, 163
 - non-vertical, 182–183
 - optimizing, for small screens, 274–277
 - orientation-specific, 276–277
 - portrait orientation, 276–277
 - scaling and resizing, 268–273
 - units, 279
 - vertical, 182
- Layout option sheet, 88
- Connected Disks page, 111
 - Bookshelf view, 112
 - Directory view, 111
 - Connected Printers page, 117
 - Network View page (Disks section), 115
 - Show list, 88
- Left Arrow gesture, 25
- Left-Down gesture, 26
- in gesture mode, 259
 - guidelines, 240

- Left-Up gesture, 26
 - in gesture mode, 259
 - guidelines, 240
- Line Height menu (MiniNote Paper sheet), 138
- Lists, 29–32, 181–188
 - boxed, 32
 - checkboxes, 32
 - checklists, 29
 - with fields, 186
 - multiple, 31
 - pop-up, 30
 - vs. boxed, 186
 - discontiguous selection in, 282–283
 - editable, 187
 - moving items in, 292
 - multiple, 184–185
 - multiple choice, 184
 - pop-up variations of, 183
 - scrolling, 186–188
 - checklists, 196
 - gesture accelerators for, 188
 - multiple, 187
 - pop-up checklists, 196
 - of selectable items, 187
 - selection feedback and, 279
 - text vs. pictures, 181–182
 - toggle switch and, 30–31
 - zero or one, 185

- Magnify Text on Screen control (MiniText), 132
- Main table of contents
 - Access Speed and, 87
 - Comments option sheet and, 87
 - option sheets and, 87
 - see also* Table of contents
- Mapping, 290
- Margins
 - gesture, 135–136
 - pop-up menu for, 58
- Mark protocol, 171–172
 - Spell, Proof, Find commands and, 172
- Markup layer, 253
 - using ink and, 255
- Markup mode, 30–31
- Marquee, in copy move operations, 68–69
- Mask. *see* Icon mask
- Memory
 - Access Speed control and, 67
 - usage, 10
- Menu buttons, 29
- Menu labels, 42
 - for current mode, 249
- Menu line, 56
 - bold buttons on, 177
- Connected Disks page, 107
 - designing, 226
 - hiding, 56
 - illustrated, 55
 - MiniNote, 136
 - MiniText, 129
 - for mode switch, 251
 - Network View page (Disks section), 114
 - table of contents, 85
 - zoom control in, 266–267
- Menus, 41–42, 192–199
 - behavior of, 42
 - checklists in, 184
 - closing, 42
 - Connected Disk page, 107
 - Connected Printers page, 116–117
 - controls in, 41–42
 - deciding when to use, 226
 - default, 56
 - dividing lines in, 197
 - fields in, 199
 - font face, 271
 - font size, 271
 - hierarchical, 42, 199
 - Installed Software section, 98
 - layout of, 196–197
 - groups of controls separation and, 197
 - multi-column menus, 196
 - for margins, 58
 - MiniNote, 136–137
 - Arrange menu, 137
 - Edit menu, 136–137
 - Pen menu, 137
 - MiniText, 129–130
 - default, 129
 - View menu, 130
 - mode switch following, 251
 - mode switch in, 251
 - for network disks, 114
 - network printers and, 119
 - Out box, 122
 - standard, 192–196
 - Document menu, 192–194
 - Edit menu, 192, 194–195
 - Options menu, 192
 - View menu, 195–196
 - strengths and weaknesses of, 225
 - submenus and, 42
 - summary, 40
 - table of contents, 85–87
 - Create menu, 86–87
 - Document menu, 85
 - Edit menu, 86
 - Sort By menu, 88
 - View menu, 86
 - two-state switches in, 198
 - View menu (Connected Disk), 108–110
- with zoom control, 266–267
 - see also specific types of menus*
- Message line, 214
- Messages
 - completion, 211
 - help, 215
 - progress, 211
 - wording of, 210
- M gesture, 27, 56
- MIL (Machine Independent Layer), 5
- MiniNote, 135–142
 - edit pads and, 47
 - gesture margin, 135–136
 - gesture mode, 135
 - gestures, 140–142, 257
 - capital letter accelerators, 142
 - gesture mode, 141–142
 - ink and gesture modes, 140
 - icon, 77, 216
 - ink mode, 135
 - menu line, 136
 - menus, 136–137
 - Arrange menu, 137
 - Edit menu, 136–137
 - Pen menu, 137
 - mode toggle, 135
 - option sheets, 138–139
 - Paper, 138–139
 - Pen, 139
 - reference buttons and, 171
- MiniText, 129–134
 - check and, 46
 - edit pads and, 47
 - embedded writing pads, 49
 - gestures, 133–134
 - icon, 77, 216
 - Insert menu, 195
 - menus, 129–130
 - option sheets, 44, 130–132, 204
 - Character, 131
 - correlation to objects, 204
 - Display, 132
 - Paragraph, 131
 - Tab Stops, 132
 - Options menu, 45
 - reference buttons, 171
 - labels, 172
 - text component and, 130
 - writing pad, 48
- Modal applications, 253
 - deferred and immediate translation, 253
 - markup layer, 253
- Modal dialog sheets, 165, 205–206
 - uses of, 206
- Modal notes, 52
- Modal objects, 43
 - distinction of, 43

- Modeless dialog sheets, 205
 - advantages of, 205
- Models, user, 245–246
 - for input modes, 246
 - selection and, 278
- Modes, 244
 - design checklist and, 295
 - dual, 248
 - explicit, 244–245
 - gesture, 245–246
 - core, 247
 - ink, 245–246
 - multiple exclusive, 248–249
 - boxed list, 249
 - pop-up list, 249
 - non-exclusive, 250
 - “safe,” 247
 - spacial, 244
 - switching, 247
 - dual command path and, 246
 - toggle gesture for, 247–248
 - see also* Input modes
- Mode switch, 247
 - location, 251–252
 - menu line, 251
 - palette line, 252
 - pop-up palette, 252
 - for markup layer, 253
 - not hiding, 250
 - presentation, 248–250
 - dual modes and, 248
 - multiple exclusive modes and, 248–249
 - non-exclusive modes and, 250
 - pictures vs. text, 248
- Modular design, 152–153
- Move command (Edit menu), 62
- Move function, 16
- Move marquee, 68, 287
- Moving
 - as adjusting, 289
 - between applications, 289
 - and copying guidelines, 285–294
 - drag & drop gesture for, 68
 - figure to far destination, 294
 - import/export and, 70
 - item in list, 292
 - in mouse-based interfaces, 285
 - in PenPoint, 286
 - variations, 289–290
 - word in text, 69, 291
 - see also* Copying
- Multiple checklists, 31
 - illustrated, 31
 - scrolling, 187
- Multiple dialog sheets, 205
- Multiple lists, 184
- Multiple undo model, 194
- Name clash notes, 71–72
 - directories and, 72
- Network disks, 114–115
 - icon, 77
 - Layout option sheet for, 115
 - Options menu for, 114
 - View menu for, 114
- Network printers, 118–119
 - menus, 119
 - option sheets, 119
- Networks, browsing, 105
- Network View page
 - Disks section, 114–115
 - Printers section, 118–119
- New technology, 9–11
 - new devices and, 10
 - new operating system and, 10–11
 - new user, 9
 - interface and, 11
- N gesture, 27
 - in gesture mode, 259
 - MiniNote, 142
 - MiniText, 134
- Non-core gestures, 23
 - application functionality and, 224
 - for gesture mode, 257, 259
 - listed, 25–27
 - MiniText, 133–134
- Non-Modal objects, 43
 - distinction of, 43
- Non-standard modeless command
 - buttons, 179
- Notebook, 14–15
 - for accessing information, 15
 - applications, 159
 - documents as one page of, 157
 - icons, 76
 - illustrated, 12
 - In box, 121
 - metaphor, 14
 - multiple, 14
 - Out box, 122
 - Stationary, 120
 - for structuring information, 14
 - table of contents, 12, 15, 85–89
 - checkboxes in, 32, 185
 - Create menu, 86–87, 195
 - Document menu, 85
 - Edit menu, 86
 - menu line, 85
 - name clash notes, 71
 - option sheets, 87–89
 - View menu, 86
 - zero or one list style, 185
 - see also* Connections notebook; Help notebook; Settings notebook; Stationary notebook
- Notebook and pen, 11
- Notebook metaphor, designing for, 151
- Notepaper ink component, 255
 - see also* MiniNote
- Notes, 52
 - completion, 211
 - confirmation, 212
 - error, 212
 - input behavior for, 43
 - name clash, 71–72
 - progress, 211
 - summary, 40
 - timing-triggered, 212–213
 - see also* specific types of notes
- Objects
 - creating, 195
 - default, 203
 - dragging, 287
 - menu commands and, 195
 - option sheet, 204
 - pages and, 46, 204
 - realistically rendered, 227
 - selection feedback for, 278–279
 - targeting, 232
 - window, 233
- Open documents, 168–169
- Open gestures, 236
 - guidelines for, 240
- Open icon, 74, 221–222
- Operating system
 - comparisons, 10–11
 - Settings notebook and, 90
- Options
 - global, 204
 - In box document, 164
 - startup, 204
- Options function, 16
- Option sheets, 44–47, 200–204
 - adding new document-wide, 202
 - adding to default document-wide, 200–202
 - bringing up, 44
 - checklists in, 184
 - clean and dirty controls, 46–47
 - Controls, 56
 - deciding when to use, 226
 - default, 200–202
 - Access sheet, 201
 - Comments sheet, 202
 - Controls sheet, 201
 - customizing, 201–202
 - design checklist and, 295
 - Disk, 112
 - Document, 89
 - document-wide, 202
 - dynamic behavior, 203–204
 - changing top sheet, 204

- relationship to selection, 203
 - response to check gesture, 203
 - global application options and, 204
 - icon, 75
 - in-line, 171
 - for keyboard accessory, 202
 - Layout, 88
 - Connected Disks page, 111–112
 - Connected Printers page, 117
 - Network Printers page, 119
 - Network View page (Disks section), 115
 - layout guidelines, 206–209
 - deactivating vs. hiding controls, 209
 - non-standard layout, 206
 - pop-up vs. in-line, 207–208
 - standard layout, 206
 - MiniNote, 138–139
 - Paper sheet, 138–139
 - Pen sheet, 139
 - MiniText, 43, 130–132
 - multi-page, 43
 - pages and object types, 46
 - pop-up lists, 183
 - printer, 118
 - Quick Help and, 215
 - reference button, 80
 - resize handles for, 272
 - response to check gesture, 46
 - section, 89
 - selection in, 283–284
 - selection relationship, 44–45
 - strengths and weaknesses of, 225
 - summary, 40
 - table of contents, 87–89
 - usage guidelines, 200
 - user model, 44
 - uses, 44
 - vertical checklists, 182
- Options menu, 44, 66–67, 192
- Access sheet, 67
 - Comments sheet, 67
 - for connected disk, 111
 - for connected printers, 116
 - Controls sheet, 60, 66
 - default commands, 56
 - document-wide option sheets and, 202
 - global and startup options and, 204
 - main table of contents, 87
 - three states of, 87
 - for MiniText application, 45
 - for network disks, 114
 - for network printers, 119
- Out box, 122
- icon, 13, 76
 - state of application and, 222
 - menus, 122
 - table of contents, 122, 160
 - for customized service, 161
 - Enabled column, 160
 - for printer service, 161
 - Status column, 160
 - transfer service and, 159–161
- Outlined buttons, 176
- Overwrite boxes, 33
- Overwrite fields, 33
- checklists with, 186
 - editing gestures for, 34
 - edit pads and, 47
 - guidelines for using, 188
 - menus with, 41, 199
 - Proof sheet and, 65
 - scaling, 269–270
 - size of, 188
 - visual segmentation cues and, 241
 - see also* Fill-in fields
-
- Pads, 47–51
- boxed/ruled, 50
 - using, 51
 - edit, pop-up, 47
 - input behavior for, 43
 - using, 50
 - writing
 - embedded, 49
 - pop-up, 48
 - summary, 40
- Page
- accessing, 15
 - control, 157
 - floating, 15
 - menu, 158
 - number, 15
 - option sheet, 46
 - turning, 15–16
- Page-oriented documents, 157–158
- Paging control, 157–158
- local page numbers and, 158
 - pop-up list on title line and, 158
- Palette construction, 184
- Palette lines, 39
- deciding when to use, 226
 - hiding, 224
 - for mode switch, 252
 - strengths and weaknesses of, 225
- Palette. *see* Accessories, palette; Pop-up palette
- Paper-like visuals, 295
- Paper option sheet (MiniNote), 138–139
- Line Height menu, 138
 - Paper Width menu, 139
- Paper Width menu (MiniNote Paper sheet), 139
- Paragraph option sheet (MiniText), 131
- Paragraph page, 46
- Pattern descriptions, 243
- PC's, comparison with pen-based computers, 10
- Pen, 152
- comparison with mouse & keyboard, 152
 - designing for, 152
 - down point, 231
 - gestures. *see* Gestures
 - ink mode picture, 248
 - input, 135
 - menu (MiniNote), 137
 - modes and, 244–245
 - option sheet (MiniNote), 139
 - preferences, 92
 - selection, 137
 - styles, 256
 - toggle switch and, 30–31
- Pen-based computers, comparison with PC's, 10
- Pen-down gestures, 231
- PenPoint
- comparison with disk-based systems, 10–11
 - direct manipulation, 151
 - hardware requirements, 5
 - In box and, 121
 - installed version, 104
 - standard editing commands, 86
 - status sheet, 104
 - zooming and, 265
- P gesture, 27, 65
- MiniNote, 142
 - MiniText, 134
- PICSEG data format, 289
- Pictures
- appropriate use of, 182
 - list of, 181–182
 - for mode switch presentation, 248
 - with text, 183
 - see also* Icons
- Pigtail gesture, 17, 24
- in gesture mode, 258
 - guidelines for, 237
 - hot point for, 231
 - MiniNote, 140
- Pixels, 153
- icon dimensions and, 217
 - icon mask and, 221
 - minimum number of, 274
 - size of, 274
- Placeholder icon, 77
- Plus gesture, 26
- for discontinuous selection, 282
 - in gesture mode, 259
 - guidelines for, 238
 - hot point for, 231
 - MiniNote, 140, 141
- Pop-up accessory applications, 159

- Pop-up edit pads, 47, 189
 - resize handles for, 272
- Pop-up lists, 183
 - multiple modes and, 249
 - scrolling, 196
- Pop-up palette
 - benefits, 252
 - for mode switch, 252
- Pop-up style (embedded document), 170–171
 - closed/open illustrations, 170
 - dialog/option sheets and, 207–208
 - mixing with in-line style, 207–208
- Pop-up writing pads, 48
 - embedded pad comparison with, 49
 - resize handles for, 272
- Portrait orientation, 276–277
- Power preferences (Settings notebook), 96–97
- Practice button (Installed Handwriting page), 100
- Preference settings, 101
- Preferences section (Settings notebook), 90, 91–98
 - Date page, 94
 - Float & Zoom page, 94
 - Fonts & Layout page, 93
 - Pen page, 92
 - Power page, 96–97
 - Sound page, 95
 - Time page, 95
 - Writing page, 92
 - pad style and, 48
 - user preference control, 48, 49
- Press gesture, 16, 24
 - family, 235
 - in gesture mode, 258
 - guidelines for, 236
 - hot point for, 231
 - for initiating drag, 286
 - for ink mode selection, 255–256
 - mapping, 290
 - MiniNote ink mode, 135
 - for zooming, 266
 - see also* Gestures
- Press Timeout, 93
- Print command, 13
 - Document menu, 57
- Printer
 - icon, 77
 - option sheet, 116, 118
- Printer Creation dialog, 117
- Printers
 - connected, 116–118
 - list of, 57
 - adding to, 163
 - network, 13, 118–119
 - services and, 163–164
 - set up, 105
- Print Format (MiniText View menu), 130
- Print Layout sheet, 58
- Print Preview command (Document menu), 193–194
- Print Setup command (Document menu), 57, 193
- Print Setup sheets, 58
 - adding, 193
 - Embedded Printing sheet, 60–61
 - Headers and Footers sheet, 59
 - Print Layout sheet, 58
- Print sheet, 57
 - modifying or replacing, 194
 - paper width and, 139
- Processor power, 96–97
- Progress note, 211
- Proof command (Edit menu), 65
 - mark protocol and, 172
- Proof sheet, 65
- Protocol
 - embedding, 166–167
 - mark, 171–172

- Quadruple-Flick, 25
- Quadruple-Tap, 25
 - in gesture mode, 259
 - MiniText, 133
- Questionmark gesture, 238
- Quick Help, 123, 215
 - applications and, 215
 - design checklist and, 295
 - sheet, 13, 123
- Quick Installer, 113
 - controls, 112, 113
- Quick Install sheet, 113

- Raised buttons, 28, 176
- RAM
 - capacity, 103
 - efficiency, 103
- Reactivating, installable software, 98
- Readability, designing for, 153
- Reference buttons, 17, 20, 79–80
 - Application Framework and, 152
 - creating, 20, 79
 - design checklist and, 296
 - gesture response of, 79
 - illustrated, 28
 - labels, 172
 - mark protocol and, 171–172
 - option sheets, 80
 - outlined style, 176
 - uses, 171
 - see also* Buttons
- Resizing, 272–273
 - handles, 272
 - minimum and maximum sizes, 272–273
 - scaling on resize, 273
- Resources, installing and deinstalling, 109
- Revert command (Document menu), 57, 192
- Right Arrow gesture, 25
- Right-Down gesture, 26
 - guidelines, 240
 - ink editing, 259
 - MiniNote, 141
 - MiniText, 134
- Right-Up-Flick gesture, 26
 - guidelines, 240
 - MiniText, 133
- Right-Up gesture, 26
 - guidelines, 240
 - ink editing, 259
 - MiniNote, 141
 - MiniText, 133
- RTF (Rich Text Format), 289
- Ruled pads, 50
 - compared to boxed pads and fill-in fields, 50
 - writing, 48
 - pop-up, 48
 - ruled, 49
 - using, 51

- Scalability
 - design checklist and, 295
 - designing for, 153–154
- Scaling, 268
 - on resize, 273
 - problems, 270
 - strategies, 268
 - with system font, 269–270
 - unscaled building blocks, 269–270
 - using layout units, 269
 - with user defined font, 271
- Scratch Out gesture, 25
 - guidelines for, 238
 - MiniNote, 140
 - MiniText, 133
- Screen Format (MiniText View menu), 130
- Screen layout, 12
- Screen size, 274
- Scribble data format, 289
- Scribbles, 254
 - Pen menu and, 137
- Scroll function, 16

- Scrolling, 16, 263–265
 - to beginning/end, 36, 264
 - by flicking, 264–265
 - by screenful, 36, 263
 - by unit, 36, 263
 - guidelines for, 263–265
 - horizontal, 274–275
 - line to top/bottom, 264
 - lists, 186–188
 - checklists, 160
 - gesture accelerators for, 188
 - multiple checklists, 187
 - pop-up checklists, 196
 - of selectable items, 187
 - marked location, 172
 - with Tap & Flick, 37
 - with thumbing, 36
- Scroll margins, 35–37, 56
 - Arrow & Drag Box style, 263–264
 - components, 36
 - control (Fonts & Layout preferences), 93
 - gestural, 37
 - hiding, 56
 - horizontal, 35
 - illustrated, 55
 - operations, 36, 263–264
 - showing and hiding, 264
 - Tap & Flick style, 37, 264
 - vertical, 35
- Search (Find & Replace sheet), 63
- Search. *see* Find command (Edit menu)
- Sections
 - Connections notebook, 105
 - creating new, 86–87
 - icon (open/closed), 76
 - option sheets, 89
 - Settings notebook, 90
- Select All command (Edit menu), 62
- Select function, 16
- Selecting, 278–284
 - deselecting and, 279–283
 - an object, 278
 - a single object, 279
- Selection
 - automatic, 283
 - current, 278
 - dual, 284
 - extending, by dragging, 280
 - extending, with brackets, 280–281
 - feedback, 187, 278–279
 - for different object types, 278–279
 - inversion and, 278
 - gestures, 187, 279
 - gestures over, 256
 - list, adjusting, 281
 - multiple, 280
 - discontiguous, 282–283
 - option sheet relationship to, 203
 - in option sheets, 283–284
 - preserved, 284
 - targeting, 232
 - text, 283
 - adjusting with brackets, 281
 - move marquee for, 287
 - user model and, 278
- Send command, 13
 - Document menu, 57
- Send Document command (table of contents Document menu), 85
- Service applications, 159
- Services, 163–165
- Settings
 - icon, 13, 76
 - powering down and, 97
- Settings notebook, 90–106
 - configuration modification and, 164
 - contents, 91
 - Fonts and Layout page
 - Field Font, 271
 - Hand Preference, 35
 - PenPoint Font, 271
 - scroll margin selection, 263
 - Tap & Flick, 37
 - Top Edge control, 276
 - Installed Software section, 90, 97–101
 - Applications page, 98
 - Installed dictionaries page, 100
 - Installed Fonts page, 101
 - Installed Handwriting page, 99–100
 - Installed Services page, 164
 - Installed User Profiles page, 101
 - organization, 90
 - overview, 90–91
 - Preferences section, 90, 91–98
 - Date page, 94
 - Float & Zoom page, 94
 - Fonts & Layout page, 93
 - Pen page, 92
 - Power page, 96–97
 - Sound page, 95
 - Time page, 95
 - Writing page, 92
 - scaling and resizing and, 268
 - Writing Timeout preference, 33
- Status section, 90, 102–104
 - PenPoint status sheet, 104
 - Storage Details sheet, 103
 - Storage Summary sheet, 102
- table of contents, 91
 - Install button, 99
- see also* Connections notebook; Help notebook; Notebook; Stationary notebook
- S gesture, 27, 64
 - MiniNote, 142
 - MiniText, 134
- Sheets
 - dialog, 40
 - input behavior for, 43
 - modal, 178
 - modeless, 178
 - multi-page, 43
 - option, 44–47
 - summary, 40
 - summary, 40
 - see also* Layout option sheet; Option sheets
- Show Special Characters control (MiniText), 132
- Shutdown button (Power preferences), 96, 97
- Signature pads, scaling, 269–270
- Small icons, 74
- Sort By menu (table of contents), 88
- Sound preferences (Settings notebook), 95
- Spacial modes, 244
- Spell checking, 64
- Spell command (Edit menu), 62, 64
 - mark protocol and, 172
- Spell sheet, 64
 - modifying and replacing, 194
- Spreadsheet, embedded chart in, 167
- Square buttons, 28, 176
- Standard modal command buttons, 178–179
- Standard modeless command buttons, 178
- Stationary, 120
 - custom, 120
 - design checklist and, 295
 - directory, 162
 - document, 162
 - icon, 13, 76
 - installable, 162
- Stationary notebook, 13, 120
 - document installation and, 162
 - for document stationary, 162
 - table of contents, 120
 - see also* Connections notebook; Help notebook; Notebook; Settings notebook
- Status gauges, 13
- Status section (Settings notebook), 90, 102–104
 - PenPoint status sheet, 104
 - Storage Details sheet, 103
 - Storage Summary sheet, 102
- Storage Details sheet (Status section, Settings notebook), 103
- Storage space, 102
 - allocation, 103

- Storage Summary sheet (Status Section, Settings notebook), 102
 - Storing, embedded objects, 166
 - Styles, button, 175–178
 - Submenus, 42
 - Subpage controls, 38
 - mode switch and, 251
 - Sub-pages, 158
 - Suspend
 - button (Power preferences), 96
 - capability, 96–97
 - Switch
 - mode, 247
 - location, 251–252
 - presentation, 248–250
 - two-state, 198
 - see also* Toggle switches
 - System
 - font
 - scaling with, 269–270
 - size of, 274
 - layout, 153
 - log icon, 77
-
- Table of contents
 - Connections notebook, 105
 - default object, 46
 - document icons in, 216–217
 - edit pads and, 47
 - embedding documents and, 20
 - Help notebook, 124
 - icons, 74
 - hot points and, 217
 - In box, 121
 - main, 87
 - menu line, 85
 - multiple documents and, 86
 - Notebook, 12, 15
 - checkboxes in, 185
 - menus, 85–87
 - option menus, 87–89
 - zero or one list style, 185
 - option sheets, 87–89
 - Document, 89
 - icon, 75
 - Layout, 88
 - Out box, 122, 160
 - for customized service, 161
 - Enabled column, 160
 - for printer service, 161
 - Status column, 160
 - Settings notebook, 91
 - Install button, 99
 - Stationary notebook, 120
 - Tables
 - discontiguous selection in, 283
 - dragging in, 288
 - horizontal scrolling and, 274–275
 - Quick Help and, 215
 - selection feedback and, 279
 - Tabs, 15
 - Show Special Characters control (MiniText), 132
 - toggling, 32
 - Tab Stop option sheet (MiniText), 132
 - Tap & Flick scroll margins, 37, 264
 - Tap gesture, 16, 24
 - Double, Triple, Quadruple, 25
 - in gesture mode, 259
 - guidelines for, 239
 - for zooming, 266
 - family, 24
 - summary, 235
 - in gesture mode, 258
 - guidelines for, 236
 - hot point for, 231
 - for ink mode selection, 255–256
 - MiniNote, 140, 141
 - MiniText, 133
 - for selection, 279
 - Tap & Flick scroll margins and, 37, 264
 - see also* Gestures
 - Tapping, 17
 - boxed pads and, 50
 - clean and dirty controls and, 46–47
 - for deselection, 280
 - for drag & drop cancellation, 69
 - edit pads and, 189
 - Help notebook, 124
 - icons, 74
 - Installed Dictionaries page, 100
 - Installed User Profile page, 101
 - in menu fields, 199
 - menus and, 42
 - MiniNote and, 135
 - multi-page sheets and, 43
 - multiple checklists, 31
 - on fill-in field, 33
 - on page number, 15
 - pop-up checklists and, 30
 - Power preferences and, 96–97
 - Proof sheet and, 65
 - Quick Help sheet, 123
 - reference buttons and, 20, 79
 - for scrolling, 36
 - for selection, 279
 - Spell sheet and, 64
 - subpage controls and, 38
 - tab, 15
 - toggle switch and, 10–181, 30
 - zero or one style and, 185
 - for zooming, 267
 - Tap-Press gesture, 16, 24
 - in gesture mode, 258
 - guidelines for, 237
 - for initiating drag, 286
 - for ink editing, 255–256, 258
 - MiniNote, 135, 140, 141
 - for zooming, 266
 - Targeting, 231–233
 - auto-selection and, 283
 - drag destination, 288
 - guidelines for, 232–233
 - selection, 232
 - unselected objects, 232
 - window objects, 233
 - zone, 232
 - Templates
 - constraining with, 242–243
 - levels of, 243
 - Text
 - boxes, 32, 33
 - component
 - displays, 233
 - MiniText and, 130
 - scaling, 269–270
 - targeting unselected objects and, 232
 - targeting zone, 232
 - default object, 46
 - dragging in, 288
 - fields, 32–34, 188
 - fill-in, 33
 - fill-in vs. overwrite, 188
 - font, 188
 - gestures in, 34
 - overwrite, 33
 - text boxes, 32, 33
 - formatting, 16
 - list, 181–182
 - for mode switch presentation, 248
 - moving word in, 69, 291
 - option sheet, 16
 - with pictures, 183
 - selection, 283
 - feedback for, 279
 - move marquee for, 287
 - T gesture, 27
 - Thumbing, 36
 - Tickmarks, for gauges, 38
 - Tidy command (MiniNote Arrange menu), 137
 - Tie gesture, 142
 - Time preferences, 95
 - Timing-triggered notes, 212–213
 - Title line, document, 56
 - capital letter gestures, 27
 - illustrated, 55
 - Quick Help and, 215
 - Toggle boarders gesture, 20
 - Toggle gesture, 247–248
 - Toggle switches, 30–31, 180–181
 - for dual modes, 248
 - operation of, 31
 - tapping, 180–181

Top Edge menu (Fonts & Layout Preferences), 93

Transfer services, 159–161
connected, 160
defined, 159
disconnected, 160
table of contents, 160–161
modifying, 161

Translate & Edit command (MiniNote Edit menu), 137

Translation
boxed pads and, 50
deferred, 253
using ink and, 255
immediate, 253
improving, 242
modes, 250
ruled pads and, 50

Translators. *see* Handwriting, translators

Triple-Flick, 25

Triple-Tap, 25
in gesture mode, 259
guidelines for using, 239
MiniNote, 141
MiniText, 133
zooming and, 266

Turning page. *see* Page, turning

Turn to command (table of contents View menu), 86

Two-state switches, 198

U gesture, 27
MiniNote, 142
MiniText, 134

UI Toolkit, 153, 205
choice list forms, 206
lists and, 181
page control, 158
pop-up writing and editing pads, 189
text field styles, 188
toggle control, 248

Underline style, 16

Undo command (Edit menu), 62, 194

Undo gesture, 16, 24
in gesture mode, 258
guidelines for, 237
hot point for, 232

Undo/redo model, 194

Units, for scrolling, 36

Up-Down gesture, 259

Up-Left gesture, 26
guidelines for, 240
in MiniNote, 142

Up-Right gesture, 26
guidelines for, 240

in MiniNote, 142
in MiniText, 133

User dictionaries, 100

User interface
access paths and, 171
automatic layout and, 153
buttons in, 151, 171
comparisons, 11
consistency in, 154
design, 149–150
consistency and, 149
development process and, 5
direct manipulation and, 151
graphic, 149
iterative development and, 151
wording and, 151
layering, 224
model, 245–246
modes and, 244–245
notebook metaphor, 151
overview, 12–20
see also UI Toolkit

User models, 245–246
drag & drop, 285–286
for input modes, 246
for moving and copying, 285–294
selection and, 278
see also Drag & drop interface

User profiles, 101
icon, 78
installed, 101
installable software views and, 109

Users, comparison of, 9

Verbs use, in button labels, 175

View menu, 196
MiniText, 130
samples of, 196
table of contents, 86
network disks, 114
network printers, 119

Visual segmentation cues, 241
spacial modality and, 244

Well-defined gestures, 236
guidelines for, 238

White space, in graphic design, 149

Window class, 233

Window objects, targeting, 233

Wipe-through, 280

Wording
of button labels, 210–211
fundamentals, 151
guidelines, 210–211
of messages, 210

Word lists, 242

Word processor
embedded icon in, text, 168
large embedded icon in, text, 168
open embedded document in, text, 169

Work area, 56
icons, 74
illustrated, 55

Work space, 102

Writing pads
boxed/ruled, 50
using, 51
embedded, 49
pop-up, 48

Writing preferences (Settings notebook), 92

Writing Timeout, 93
preference, 33

Zero or one, list style, 185

Zoom commands, 267

Zoom control, 267

Zooming, 265–267
gestures, 266
operations, 265

PenPoint™ User Interface Design Reference

PenPoint User Interface Design Reference describes the unique PenPoint Notebook User Interface (NUI) and how to use its elements in your applications. The intuitive pen-and-paper interface and its related menus, option sheets, and controls is the result of several years of design and user research. The PenPoint Software Development Kit (SDK) contains a toolkit of user interface elements that helps you build an interface that is consistent with the PenPoint NUI.

The *PenPoint User Interface Design Reference* contains two parts. The first part takes you on a tour of the user interface elements and describes where and how you should use them. The second part describes how PenPoint uses the user interface elements. If these elements are used consistently, users can apply their understanding of PenPoint across all applications.

The PenPoint operating system is a compact, 32-bit, fully object-oriented, multitasking operating system designed expressly for mobile pen computers. GO Corporation designed the PenPoint operating system as a productivity tool for people who may never have used computers before, including salespeople, service technicians, managers and executives, field engineers, insurance agents and adjustors, and government inspectors.

Other volumes in the GO Technical Library are:

PenPoint Application Writing Guide provides a tutorial on writing PenPoint applications, including many coding samples.

PenPoint Development Tools describes the environment for developing, debugging, and testing PenPoint applications.

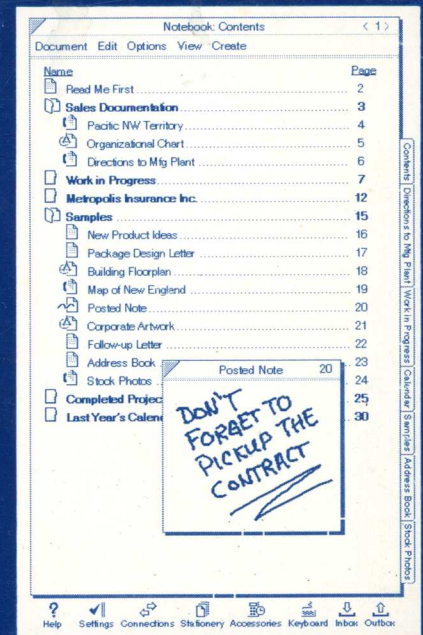
PenPoint Architectural Reference, Volume I presents the concepts of the fundamental PenPoint classes.

PenPoint Architectural Reference, Volume II presents the concepts of the supplemental PenPoint classes.

PenPoint API Reference, Volume I provides a complete reference to the fundamental PenPoint classes, messages, and data structures.

PenPoint API Reference, Volume II provides a complete reference to the supplemental PenPoint classes, messages, and data structures.

GO Corporation was founded in September 1987 and is a leader in pen computing technology for mobile professionals. The company's mission is to expand the accessibility and utility of computers by establishing its pen-based operating system as a standard.



919 East Hillsdale Blvd.
Suite 400
Foster City, CA 94404